Universidade de Aveiro
2014

Departamento de Eletrónica, Telecomunicações e Informática

RITA ALEXANDRA DA FONSECA JESUS

DINAMIZAÇÃO E DIVULGAÇÃO DA PLATAFORMA API – ACADEMIC PLAYGROUND & INNOVATION

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação científica do Doutor Cláudio Teixeira, Professor Equiparado a Investigador Auxiliar e do Doutor Diogo Gomes, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico estre trabalho aos meus pais e irmã pelo apoio incondicional e por me ensinarem a enfrentar a vida com um sorriso independentemente de todas as dificuldades encontradas.

O júri

Presidente	Doutor José Manuel Matos Moreira Professor Auxiliar, Universidade de Aveiro
Vogal – Arguente Principal	Doutora Mariana Curado Malta Professora Adjunta, Instituto Superior de Contabilidade e Administração do Porto
Vogal – Orientador	Doutor Cláudio Jorge Vieira Teixeira Professor Equiparado a Investigador Auxiliar, Universidade de Aveiro

Agradecimentos

Aos professores Cláudio Jorge Vieira Teixeira e Diogo Nuno Pereira Gomes, orientadores desta dissertação, por todos os recursos disponibilizados e pela ajuda prestada ao longo do projeto, principalmente com a reestruturação dos objetivos e encaminhamento dos mesmos.

Aos meus pais e irmã por nunca deixarem de acreditar em mim e me darem incentivo desde o primeiro até ao último momento. A vocês, agradeço por tudo o que já consegui alcançar na vida e pela pessoa em que me tornei. Sei que sem a vossa força não teria conseguido terminar este percurso de uma forma tão positiva e continuarei a contar convosco para tomar as melhores decisões para o meu futuro.

À Joana Silva e à Joana Coelho por estarem sempre presentes nos momentos marcantes deste percurso académico e da minha vida pessoal. Sem dúvida que posso dizer que termino este curso com mais duas irmãs. Tenho a certeza que sem vocês não teria chegado tão longe, o vosso apoio foi fundamental.

À Ana Marques, ao Rui Costa e ao Tiago Silva por estarem do meu lado há quase uma década e nunca deixarem de se preocupar comigo, vocês são a prova de que existem amizades que resistem a tudo, independentemente da distância e do tempo que passa.

Ao David Ferreira por tudo o que me ensinou desde que nos conhecemos e por todo o encorajamento e cumplicidade, sei que posso contar contigo para a vida. Consegues ter confiança e acreditar em mim mais do que eu própria e isso foi uma força para nunca desistir.

À Sofia Duarte, à Marta Esteves e à Cristina João por todo o apoio dado ao longo destes últimos meses e também por terem conseguido lidar com todas as minhas preocupações e dúvidas dia após dia. É inquestionável a importância da vossa ajuda aos mais diversos níveis.

Palavras-chave

Academic Playground & Innovation, serviços web, testes unitários, controlo de acessos.

Resumo

Na Universidade de Aveiro existe um portal que oferece serviços web ou APIs para utilização pelos estudantes durante o desenvolvimento de projetos académicos. Apesar da sua utilidade continua a existir uma baixa utilização desta plataforma designada por Academic Playground & Innovation. Um dos principais objetivos era divulgar e promover a mesma por forma a aumentar a sua utilização. Para isso foi estruturada uma sequência de concursos a pôr em prática e o desenvolvimento de componentes relacionados. Este trabalho foca também a temática dos webservices, passando por desenvolver e documentar novos serviços, assim como implementar testes unitários sobre webservices existentes. Importa ainda mencionar o desenvolvimento de um componente relacionado com o controlo de acessos em edifícios da academia.

Keywords Academic Playground & Innovation, webservices, unit testing, access control.

Abstract In University of Aveiro there is a website that offers web services and APIs for usage by students during the development of academic projects. Despite its usefulness, there is still a low utilization of this platform designated by Academic Playground & Innovation. One of the main goals was to outreach and promote this platform in order to increase its usage. With that in mind a sequence of contests was structured as well as the development of related components. The theme of web services is also focused on this work through developing and documenting new services, followed by the implementation of unit tests on existing web services. It is also important to mention the development of a component related with access control in buildings of the academy.

Índice

Índice de Ilustrações

1. Introdução

Antes da existência de uma utilização ativa dos computadores nas organizações, os sistemas de informação consistiam basicamente na gestão de informação através do uso de arquivos. Este método permitia à organização fazer um controlo da informação útil mas não era eficaz uma vez que era um processo manual e não possibilitava o relacionamento automático dos dados nem uma fácil manutenção dos mesmos. **[1]**

Com o passar do tempo e a evolução da tecnologia, os sistemas de informação começaram a ser desenvolvidos a nível computacional e a permitir uma gestão muito mais rápida e eficiente dos dados. Este método passou também a permitir que a informação fosse mais facilmente relacionada entre si e que não existissem tantos erros associados ao controlo e gestão dos dados existentes.

Para que fosse possível armazenar a informação necessária, surgiu o conceito de bases de dados que são estruturas capazes de guardar e organizar grandes conjuntos de dados. Estas estruturas têm associados, habitualmente, sistemas de gestão para garantir uma melhor administração dos dados existentes. **[2]**

Assim, é possível verificar que a utilização de bases de dados é uma necessidade comum à grande maioria dos sistemas de informação desenvolvidos com o intuito de possibilitar o armazenamento de informação para posterior gestão e consulta. Por vezes, é necessária a utilização de dados provenientes de diferentes fontes, algumas delas pertencentes a entidades externas.

1.1. Motivação

O fornecimento de dados a outras entidades é sempre um processo a ter em atenção a nível de segurança da informação, uma vez que podem existir utilizadores menos bem-intencionados que se acederem a informação além da pretendida podem fazer um mau uso da mesma ou provocar danos incorrigíveis.

Para prevenir a ocorrência de situações em que a informação existente é comprometida ou eliminada devido a ações de utilizadores externos, é necessária a adoção de algumas medidas. Uma delas pode passar essencialmente pelo desenvolvimento de uma camada intermédia que faz a comunicação entre estes utilizadores e a base de dados.

Para este fim podem ser utilizados *webservices* que não são mais do que um método de comunicação através de uma rede entre dois pontos específicos – que podem ser sistemas de informação, aplicações, bases de dados, … **[3]** Os *webservices* funcionam através da chamada de funções, cada uma delas associada a um fim particular. Desta forma, como a base de dados não é acedida diretamente e como não podem ser feitas ações para além daquelas que estão estipuladas, garante-se uma maior integridade e segurança dos dados existentes.

1.2. Problema

Para além da comunicação com as bases de dados, os *webservices* têm também outros objetivos de grande importância. Dependendo do contexto, a utilização de *webservices* nos sistemas de informação também pode permitir fazer uma agregação de conteúdos dentro da mesma temática mas originários de diferentes fontes fornecedoras de dados. Estes dados podem ser relacionados entre si para apresentar informação mais completa e com maior nível de detalhe.

Dentro deste contexto existe na Universidade de Aveiro uma plataforma que disponibiliza *webservices*, principalmente para utilização por parte dos alunos no âmbito da realização de projetos de diversas unidades curriculares. Esta plataforma, denominada por *Academic Playground & Innovation*, potencia a formação prática dos alunos uma vez que lhes permite aplicar os conceitos teóricos aprendidos nas unidades curriculares relacionadas com este tema, sendo assim uma mais-valia. **[4]**

Contudo, apesar de este portal ser de bastante utilidade a vários níveis, existe ainda uma baixa utilização do mesmo, sendo conhecido apenas por uma percentagem dos alunos do Departamento de Eletrónica, Telecomunicações e Informática. Para combater este facto, surgiu a ideia de divulgar na academia e dinamizar a plataforma existente.

1.3. Contextualização

Como referido anteriormente, a plataforma API – *Academic Playground & Innovation* – tem como principal objetivo disponibilizar aos alunos *webservices* com diversos tipos de conteúdos. Esta plataforma está organizada de forma a permitir uma fácil consulta da lista completa de *webservices* existentes para utilização por parte dos alunos e estruturada de modo a garantir uma rápida identificação dos *webservices* por categorias.

Se de entre a lista completa de *webservices* existentes for selecionado um em específico, é mostrada uma pequena descrição sobre o mesmo, indicando o autor, data de publicação e ainda *tags* com as categorias a que pertence. Mais importante que isso, assim que é selecionado um *webservice* particular é mostrada a sua documentação que apresenta quais as funcionalidades disponíveis e qual a correta forma de utilização das mesmas. Numa boa parte dos casos vão sendo apresentados exemplos de utilização.

Este portal encontra-se disponível publicamente no endereço `http://api.web.ua.pt` **[4]** e podem ver-se imagens relativas à sua estrutura, apresentação e conteúdos no `Anexo A – Conteúdos do portal Academic Playground & Innovation`.

Associado à plataforma API encontra-se o portal de serviços da Universidade de Aveiro que contém alguns dos *webservices* documentados nesta plataforma. Para cada serviço existente neste portal é apresentado o símbolo da *wiki* que representa uma ligação para a página de documentação presente na plataforma API. Este portal também se encontra disponível publicamente, no endereço `http://services.web.ua.pt` **[5]** e podem ver-se imagens relativas aos seus conteúdos no `Anexo B – Conteúdos do portal Serviços da Universidade de Aveiro`.

1.4. Estruturação

Os capítulos seguintes pretendem descrever cada um dos pontos desenvolvidos ao longo da dissertação, começando pelo capítulo 2 – `"Plataforma Academic Playground & Innovation"` – que indica o trabalho realizado anteriormente no âmbito deste portal e de que forma podem ser recolhidas informações importantes para a divulgação e dinamização deste, através da realização de concursos e do desenvolvimento de componentes.

Como a base fundamental da plataforma API assenta na disponibilização de serviços web, o capítulo 3 – `"Desenvolvimento de um webservice de assinaturas"` – descreve todo o processo de implementação e documentação de um *webservice* para assinaturas, mostrando também um exemplo prático da sua utilização.

Segue-se o capítulo 4 – `"Implementação de testes unitários"` – que mostra o estudo e comparação de algumas ferramentas para integração contínua e implementação de testes unitários, indicando características para cada uma delas e os motivos para escolha de uma ferramenta específica. Este capítulo descreve ainda os testes realizados sobre vários serviços web desenvolvidos.

Tendo em conta que a UA é uma instituição de ensino, em cada um dos seus edifícios existem conteúdos com valores elevados, tanto a nível de materiais como de informação, sendo a sua segurança um fator de extrema importância. Desta forma, existem em todos os edifícios dispositivos que registam e controlam todos os acessos através de cartões. No capítulo 5 – "Componente para gestão de perfis de acesso" – é feita uma análise do sistema de controlo de acessos atual e expostas as alterações necessárias para que a atribuição de perfis de acesso seja melhorada. É ainda descrito o desenvolvimento de uma interface web para interagir com uma base de dados associada ao sistema de controlo de acessos com dados de utilizadores reais da Universidade de Aveiro.

2. Plataforma Academic Playground & Innovation

2.1. Introdução

A plataforma *Academic Playground & Innovation* tem como uma das principais finalidades a disponibilização de dados aos alunos para uso em contexto académico através do fornecimento de *webservices*. A utilização destes serviços web no desenvolvimento dos seus projetos permite o contacto com diferentes fornecedores de dados e possibilita a integração e agregação de diversos tipos de conteúdos na mesma aplicação ou sistema de informação para que estes fiquem mais completos e dinâmicos dentro do contexto em que se encontram.

2.2. Trabalho anterior

Os primeiros contactos com a plataforma API e o portal de serviços da Universidade de Aveiro foram enquanto aluna da Licenciatura em Tecnologias e Sistemas de Informação. Neste percurso tive a oportunidade de conhecer os objetivos de ambas as plataformas e de utilizar a nível académico alguns dos serviços disponibilizados pelas mesmas. Isto também aconteceu em algumas unidades curriculares enquanto aluna do Mestrado em Sistemas de Informação.

Contudo, o contacto mais próximo que tive com a *Academic Playground & Innovation* foi iniciado em Fevereiro de 2013 no âmbito de uma Bolsa de Integração na Investigação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro. Esta bolsa tinha como principais finalidades a gestão e manutenção de serviços web já existentes, o desenvolvimento e documentação de novos *webservices* e a identificação de serviços com necessidade de revalidação. Uma vez que os serviços web desenvolvidos a nível da referida bolsa serão utilizados numa das fases da presente dissertação, estes encontram-se resumidos de forma breve nos próximos pontos.

2.2.1. Webservice de parques

O *webservice* de parques disponibiliza em formato JSON informação sobre a lotação dos parques de estacionamento da Universidade de Aveiro e inclui dados como a capacidade, o número de lugares ocupados, o número de lugares livres e a sua localização.

Este serviço está acessível em `http://services.web.ua.pt/parques/parques` e com a documentação associada a cada funcionalidade do mesmo disponível no endereço `http://api.web.ua.pt/pt/services/universidade_de_aveiro/lotacao_parques`.
As funcionalidades permitidas pelo referido serviço são as seguintes:

Listar todos os parques

- Retorna os dados relativos a todos os parques de estacionamento.

Dados de um parque

- Devolve todos os dados referentes a um determinado parque.

Dados de vários parques

- Devolve todos os dados referentes a vários parques especificados.

Parques por disponibilidade

- Devolve os dados relativos aos vários parques para aquela disponibilidade.

Parques por ID e disponibilidade

- Retorna dados referentes ao conjunto parques/disponibilidade especificado.

Como exemplo de funcionamento, podemos ver abaixo na `Imagem 1` o resultado em JSON retornado quando é invocada a operação de listagem de parques.


```
[{"Timestamp":1368692695},
{"ID":"P1","Nome":"Publico","Capacidade":663,"Ocupado":303,"Livre":360},
{"ID":"P5","Nome":"Cantina","Capacidade":249,"Ocupado":189,"Livre":60},
{"ID":"P6","Nome":"ZTC","Capacidade":10,"Ocupado":35,"Livre":-25},
{"ID":"P8","Nome":"Subterraneo","Capacidade":74,"Ocupado":4,"Livre":70},
{"ID":"P9","Nome":"Ceramica","Capacidade":49,"Ocupado":22,"Livre":27},
{"ID":"P10","Nome":"Edificio 2 e 3","Capacidade":36,"Ocupado":39,"Livre":-3},
{"ID":"P11","Nome":"Edificio 1","Capacidade":72,"Ocupado":57,"Livre":15},
{"ID":"P13","Nome":"ISCAA Publico","Capacidade":95,"Ocupado":6,"Livre":89},
{"ID":"P14","Nome":"ISCAA Funcionarios","Capacidade":46,"Ocupado":0,"Livre":46},
{"ID":"P15","Nome":"ESTGA","Capacidade":188,"Ocupado":48,"Livre":140}]
```


Imagem 1 – Parques: listagem de parques de estacionamento

2.2.2. Webservice de edifícios

O *webservice* de edifícios disponibiliza em formato JSON informação sobre os edifícios e unidades da Universidade de Aveiro e inclui dados como coordenadas geográficas, pisos de cada edifício, infraestruturas e salas existentes em cada piso de um edifício, entre outras informações.

Este serviço está acessível em `http://services.web.ua.pt/arcgis/arcgis` e com a documentação associada a cada funcionalidade do mesmo disponível no endereço `http://api.web.ua.pt/pt/services/universidade_de_aveiro/edificios_ua`. As funcionalidades permitidas pelo referido serviço são as seguintes:

Listar *endpoints*

- Apresenta uma lista com os diferentes *endpoints* e respetiva indicação do edifício ou unidade que representam.

Listar edifícios

- Retorna uma listagem com dados relativos a todos os edifícios.

Listar unidades

- Devolve uma lista com informações associadas a todas as unidades.

Listar pisos

- Apresenta uma lista com os dados relativos aos pisos de um determinado edifício.

Listar infraestruturas

- Retorna uma listagem com as infraestruturas de cada piso de um edifício específico.

Listar salas

- Devolve uma lista com os dados associados às salas de um edifício em particular.

Listar elementos

- Apresenta uma lista com todos os tipos de elementos existentes em cada piso de um determinado edifício.

Listar mapas

- Mostra uma listagem de endereços que permitem visualizar mapas de pisos e infraestruturas de um edifício específico.

Como exemplo de funcionamento, podemos ver abaixo na `Imagem 2` o resultado em JSON retornado por este *webservice* quando é invocada a operação de listagem de mapas do departamento de eletrónica, telecomunicações e informática.

```
{
    "edificio": "ed4/electronica",
    "pisos": [
        {
            "ID": 0,
            "nome": "Piso 1",
            "infraestruturas": null,
            "mapa":
"http://websig.ua.pt/ArcGIS/rest/services/ed4/electronica/MapServer/export?
f=image&format=png&transparent=true&size=1000%2C600&layers=show%3A14"
        },
        {
            "ID": 15,
            "nome": "Piso 2",
            "infraestruturas": null,
            "mapa":
"http://websig.ua.pt/ArcGIS/rest/services/ed4/electronica/MapServer/export?
f=image&format=png&transparent=true&size=1000%2C600&layers=show%3A29"
        },
        {
            "ID": 30,
            "nome": "Piso 3",
            "infraestruturas": null,
            "mapa":
"http://websig.ua.pt/ArcGIS/rest/services/ed4/electronica/MapServer/export?
f=image&format=png&transparent=true&size=1000%2C600&layers=show%3A43"
        },
        {
            "ID": 44,
            "nome": "Cobertura",
            "infraestruturas": null,
            "mapa":
"http://websig.ua.pt/ArcGIS/rest/services/ed4/electronica/MapServer/export?
f=image&format=png&transparent=true&size=1000%2C600&layers=show%3A56"
        }
    ]
}
```


Imagem 2 – Edifícios: listagem de mapas do DETI

2.3. Público-alvo

Neste momento, os conteúdos existentes na plataforma API direcionam-se de uma forma particular para os alunos do Departamento de Eletrónica, Telecomunicações e Informática visto possibilitarem maior criação de valor a nível pedagógico no contexto dos cursos deste departamento em comparação com os restantes cursos da Universidade.

Apesar disso, as características desta plataforma podem ser também atraentes aos alunos dos restantes departamentos, bastando para isso fazer o fornecimento de conteúdos de interesse a cada curso em particular. Assim sendo, um dos objetivos passa por perceber quais as necessidades associadas a cada curso a nível de disponibilização de informação que possa ser feita através de serviços web.

Dentro de cada departamento existe uma quantidade gigantesca de informação inserida em documentos, bases de dados, sistemas de informação, etc. A grande maioria dessa informação é, naturalmente, privada e não necessita nem deve ser acedida por mais do que um pequeno conjunto de pessoas.

Contudo, existem dados que poderiam ser tornados públicos e que enriqueceriam muito o conhecimento dos alunos dando-lhes a oportunidade de tomar conhecimento dos mesmos e utilizá-los em âmbito académico.

Para que isso fosse possível era necessária a intervenção de membros dos diferentes cursos e departamentos da academia para que fosse assim permitido averiguar quais as necessidades existentes, de forma a arranjar depois um meio para colmatar as falhas de informação reportadas pelos mesmos.

2.4. Divulgação da plataforma

Considerando que um dos principais objetivos desta dissertação era fazer a dinamização da plataforma API, era necessário fazer primeiro uma pequena divulgação que desse a conhecer esta plataforma aos estudantes da UA. Para isso foi desenhado um *flyer* em formato digital que explicava o conceito do portal API e fazia uma pequena introdução ao objetivo de um concurso de ideias a realizar no âmbito da dinamização da plataforma – a ser descrito mais à frente.

O referido folheto, disponível no `Anexo C – Flyer enviado para divulgação da plataforma API`, foi anexado a um email enviado à secretaria do departamento que se encarregou de o reencaminhar para as restantes secretarias departamentais da UA. Cada secretaria, por sua vez, tratou da divulgação interna do mesmo pelos alunos do seu departamento. Desta forma, a informação a ser divulgada acerca da plataforma API e de um dos concursos associados à sua dinamização chegou a todos os alunos.

Com vista a uma dinamização estruturada foi pensada uma série de concursos a pôr em prática após a divulgação da plataforma. Estes concursos serão descritos no `capítulo 2.5 – Dinamização da Plataforma` e tinham em comum a forte aposta na diversidade de participações, envolvendo sempre alguma forma os alunos dos vários cursos e departamentos.

2.5. Dinamização da plataforma

2.5.1. Concurso de ideias

No folheto de divulgação da plataforma é referido que a associação de conjuntos de dados à plataforma API permite facilitar o fornecimento de dados de forma a poder ser feita a sua visualização em diferentes cenários. É possível tomar como prova de conceito o facto de a disponibilização dos dados das ementas das cantinas nesta plataforma através de um serviço ter possibilitado o desenvolvimento de uma aplicação para consulta das mesmas em dispositivos móveis.

O primeiro concurso da sequência prevista para dinamização do portal era o concurso de ideias, cujo objetivo era entender quais os dados existentes em bases de dados internas aos departamentos que pudessem tornar-se de acesso público devido à associação à plataforma API.

Assim que existisse uma boa quantidade de ideias provenientes dos vários departamentos seriam começados a estabelecer contactos com algumas empresas dentro das diferentes áreas de ação para que estas fossem parte integrante do concurso, tanto a nível de oportunidades como a nível de contribuição com ideias ou avaliação das ideias existentes. Para uma melhor contextualização das empresas no âmbito do projeto seria feita uma introdução à plataforma em si e especificados quais os principais objetivos da presente dissertação e do concurso de ideias em concreto.

Exemplo de ideia

Nesta fase, cada aluno interessado em contribuir com ideias deveria indicar que tipo de dados poderiam ser úteis de disponibilizar em serviço web, podendo ser provenientes de diversas fontes. Exemplos disso são os dados relativos a estudos sobre qualquer tema de interesse, bem como informação associada a análises ou experiências feitas por docentes, investigadores, ou alunos em âmbito académico.

O Clime@UA **[6]** é um *website* pertencente ao Grupo de Meteorologia e Climatologia da Universidade de Aveiro que apresenta previsões em diversos pontos do país referentes a dados meteorológicos, marítimos e oceanográficos. Este *website* apresenta uma grande utilidade pois permite consultar, entre muitas outras coisas, dados de temperatura, vento e precipitação em forma de gráficos. É possível ver na `Imagem 3` o exemplo de um gráfico que mostra previsões de temperatura para uma semana em Aveiro.

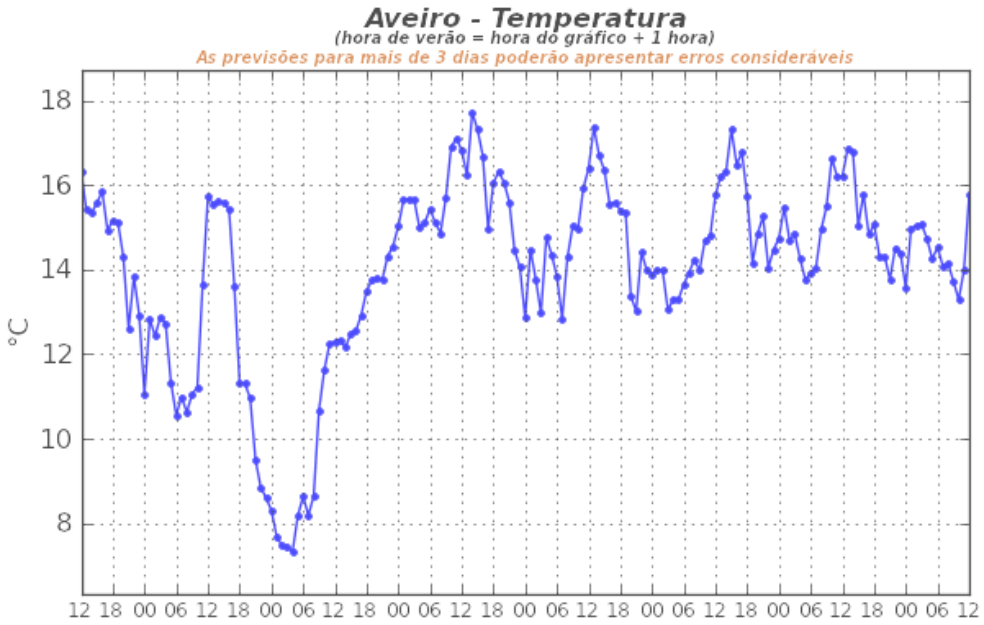

Imagem 3 – Clim@UA: exemplo de gráfico para a temperatura em Aveiro (1 semana)

Considerando este portal de previsões como exemplo, poderia ser sugerida como ideia para o concurso a implementação de um *webservice* que fizesse a disponibilização dos dados inerentes ao mesmo. Uma vez que o fornecimento de dados referentes a previsões futuras poderia apresentar inconvenientes, poderia ser sugerido o envio apenas dos dados relativos a previsões efetuadas até à data atual (ou dia anterior) para uso, por exemplo, em projetos de alunos do curso de Meteorologia, Oceanografia e Geofísica.

2.5.2. Concurso de webservices

A grande finalidade do concurso de ideias descrito anteriormente era a implementação das ideias propostas pelos participantes. Após a análise das mesmas seria elaborada uma lista com todas as propostas concretizáveis e, caso essa lista apresentasse um baixo número de propostas, seriam desenvolvidas no âmbito desta dissertação.

Considerando a possibilidade de o número de propostas ser extenso ao ponto de não ser possível implementar todos os *webservices* em tempo útil, seria organizado um concurso de *webservices* no seguimento do concurso anterior. Tendo em conta as várias áreas de estudos dos alunos da UA, existia a probabilidade de a maior parte dos participantes neste concurso serem alunos do Departamento de Eletrónica, Telecomunicações e Informática. Desta forma, o concurso poderia ser realizado em parceria com os núcleos de estudantes do DETI.

O objetivo principal deste concurso seria o desenvolvimento de serviços web que fossem acrescentar valor com a sua existência, isto é, que utilizassem uma ou várias das ideias propostas e possibilitassem posteriormente a visualização dos dados envolvidos em vários contextos – por exemplo sistemas de informação e aplicações móveis.

Exemplo de webservice

Nesta fase, cada aluno interessado na implementação de *webservices* deveria fazer uma descrição da finalidade do *webservice* que pretendia desenvolver, referindo quais as ideias a utilizar e as funcionalidades que estariam presentes no mesmo. Após serem submetidas as propostas, estas seriam analisadas por forma a confirmar a possibilidade de realização das mesmas e tentar que não existissem ideias muito semelhantes a ser desenvolvidas em simultâneo. Para isso, as propostas deveriam ser descritas de forma detalhada e clara para que não existissem dúvidas dos objetivos de cada uma delas.

Para apresentar um exemplo de serviço web a propor, considerou-se novamente o portal Clime@UA **[6]** do Grupo de Meteorologia e Climatologia da Universidade de Aveiro, já referido anteriormente como sendo um portal de previsões (meteorológicas, marítimas e oceanográficas). Ao entrar no portal, a página inicial apresenta um mapa do país com os diversos pontos para os quais são feitas previsões de tempo, mostrando também o valor da temperatura mínima e máxima previstos. Uma funcionalidade que o *webservice* poderia ter implementada seria a disponibilização de uma lista das cidades abrangidas por essas previsões, juntamente com os respetivos valores de temperaturas. É possível ver essa informação em formato gráfico – tal como é apresentada no *website* – na `Imagem 4`.

Imagem 4 – Clim@UA: mapa do país com as cidades abrangidas pelas previsões

Relativamente às previsões meteorológicas, após selecionada uma localidade no mapa é possível consultar, em forma de gráficos, os dados referentes a valores de temperatura, precipitação total, vento, nuvens, nevoeiro, entre outros. Associada a cada uma destas consultas poderia ser implementada uma função do serviço web que devolvesse os valores previstos para uma determinada localidade. As localidades poderiam ser identificadas por IDs que seriam listados conjuntamente com os restantes dados na função sugerida para listagem de localidades.

2.5.3. Concurso de aplicações

A grande finalidade do concurso de *webservices* era a implementação dos mesmos para que fosse possível o desenvolvimento de sistemas de informação ou aplicações móveis que os utilizassem. Isto permitiria uma maior disponibilização de informação referente às diversas áreas de estudos abrangidas pela UA.

Assim sendo, seria cumprido um dos principais objetivos da dinamização da plataforma *Academic Playground & Innovation*: colmatar algumas das falhas de informação existente que poderiam ser solucionadas com a construção de serviços web, uma vez que existem diversos dados internos aos vários departamentos que apenas não são acedidos pelos alunos devido à inexistência de meios que o possibilitem.

Após a apresentação de ideias e a implementação de *webservices* que disponibilizem a nível público dados para utilização por parte da comunidade académica, o objetivo seria a organização de um concurso para construção de aplicações móveis que utilizassem os serviços desenvolvidos para este fim. Nos últimos anos têm existido anualmente a realização de um concurso de computação móvel cuja organização está a cargo da Associação de Eletrónica, Telecomunicações e Telemática da Universidade de Aveiro – AETTUA. Posto isto, poderia ser realizada uma colaboração com esta associação do DETI para incluir os *webservices* desenvolvidos na temática da edição seguinte deste concurso.

Exemplo de aplicação

Para ilustrar um exemplo de aplicação possível considerou-se, mais uma vez, o portal de previsões Clim@UA **[6]** do Grupo de Meteorologia e Climatologia, constituído por docentes e bolseiros do Departamento de Física da UA e integrado no CESAM – Centro de Estudos do Ambiente e do Mar.

O serviço web para implementação referido no âmbito do concurso de *webservices* tinha como objetivo fornecer os dados apresentados neste portal de forma a poderem ser utilizados em outros contextos. Uma vez que já existe um sistema de informação a fazer uso desses dados – o próprio portal Clim@UA – um bom exemplo de aplicação poderia ser o desenvolvimento de uma aplicação móvel que fizesse a apresentação das informações mais importantes relativas às previsões meteorológicas.

É, sem dúvida, de grande utilidade a existência de um portal que nos permita analisar as condições meteorológicas para os dias seguintes. Ainda mais útil poderia ser a existência de uma aplicação que possibilitasse a rápida consulta dessas previsões meteorológicas através de qualquer dispositivo móvel.

Uma vez que os dados existentes são apresentados com intervalos de tempo de seis horas, esta aplicação poderia vir a ser utilizada em diferentes contextos, podendo ir desde a consulta do tempo para a hora de regresso a casa até à consulta dos dados para a manhã do dia seguinte de modo a ser possível prepara-lo de acordo com as condições climatéricas previstas.

2.6. Desenvolvimento de componentes para o portal

Ao longo dos pontos anteriores deste capítulo foram sendo descritos vários concursos a organizar para promover a dinamização da plataforma *Academic Playground & Innovation*, de modo a que passasse a ser utilizada por uma percentagem mais abrangente de alunos da Universidade de Aveiro. Em cada um desses concursos existe a necessidade de haver interação entre diversos conjuntos de pessoas e, para simplificar o contacto entre todos os envolvidos neste processo, seriam desenvolvidos alguns módulos para o portal.

2.6.1. Componentes no âmbito do concurso de ideias

Como referido anteriormente, o concurso de ideias tinha como objetivo entender quais os dados existentes em bases de dados internas aos departamentos que pudessem tornar-se de acesso público. Para simplificar o processo de análise e consulta das ideias enviadas, seria acrescentada uma secção ao portal onde as mesmas seriam inseridas juntamente com todos os detalhes enviados nas propostas. A estas publicações poderiam ser feitos comentários que contivessem, por exemplo, sugestões adicionais a cada ideia.

Essas ideias seriam transmitidas a empresas dentro das diferentes áreas de estudo para que estas tivessem possibilidade de ser parte integrante do concurso, oferecendo novas ideias ou avaliando as já existentes. Assim sendo, cada empresa teria credenciais de acesso à página em questão para que fosse possível estabelecer um contacto direto com os participantes do concurso. Utilizando essas credenciais, as empresas poderiam também fazer a introdução de novas ideias na página existente ou ainda acrescentar notas às ideias já apresentadas.

2.6.2. Componentes no âmbito do concurso de webservices

Foi mencionado previamente que a maior finalidade do concurso de *webservices* era aproveitar as ideias propostas no concurso anterior para desenvolvimento de todas as que fossem concretizáveis. À semelhança do que foi descrito para o concurso de ideias, seria acrescentada uma secção à plataforma onde seria possível introduzir as propostas de serviços web enviadas e todas as informações a si inerentes. Seria permitida a adição de comentários pelos utilizadores com o objetivo de serem propostas novas funcionalidades para os mesmos ou sugeridas alterações a fazer às funções já existentes.

Uma vez que a probabilidade de a maior parte dos participantes neste concurso serem alunos do DETI era elevada devido à sua área de estudos, este seria um concurso a realizar com a colaboração dos núcleos de estudantes deste departamento. Posto isto, seria importante a existência de um local onde os diferentes núcleos pudessem contactar os participantes e discutir ideias referentes às suas propostas ou onde pudessem ser definidos tópicos importantes relativos à organização do evento entre si.

2.6.3. Componentes no âmbito do concurso de aplicações

Tal como foi indicado, o principal foco do concurso de aplicações passava pelo uso dos *webservices* implementados no concurso respetivo de forma a permitir a visualização dos dados inerentes aos mesmos em diferentes contextos – principalmente aplicações móveis ou sistemas de informação. De forma idêntica ao processo descrito para os outros dois concursos, seria acrescentada uma secção ao portal onde as propostas de aplicações para os serviços web seriam colocadas. Seguindo a lógica anterior, poderiam também ser feitos comentários a estas publicações para adicionar novas sugestões de funcionalidades ou propor alterações às funções já existentes.

Como tem existido anualmente um concurso de computação móvel a cargo da AETTUA, poderia ser desenvolvida uma parceria com esta associação, de modo a incluir os serviços web construídos no tema da edição seguinte do concurso. Desta forma, era necessário a existência de um local onde esta associação pudesse contactar os participantes sobre os temas ligados às suas sugestões de aplicações a desenvolver ou onde fosse possível ver e definir detalhes relacionados com a estruturação do evento.

2.7. Considerações

2.7.1. Problemas encontrados

A estratégia pensada para dinamizar o portal *Academic Playground & Innovation* assentava fortemente na organização de um conjunto estruturado de concursos que desse origem ao fornecimento de conteúdos de diversas temáticas com grande utilidade a cursos de outros departamentos para além do DETI. Nesse sentido, a base para todo este processo de dinamização era o concurso de ideias que teria como resultado um conjunto de *webservices* sugeridos para serem desenvolvidos e que, posteriormente, permitiriam a utilização de diversos conjuntos de dados em sistemas de informação e aplicações móveis.

No subcapítulo referente à divulgação da plataforma foi referido o envio de um email com destino a todos os alunos dos diferentes cursos da academia, cuja finalidade era informar os alunos acerca da existência da plataforma API e do concurso de ideias para que os mesmos pudessem enviar propostas de ideias. Contudo, apesar dessa divulgação não foi recebido qualquer contacto por parte dos alunos, não sendo possível realizar o concurso de ideias por falta de propostas. Posto isto, ficou assim impedida a restante sequência de concursos previstos e o desenvolvimento de componentes inerentes aos mesmos.

2.7.2. Objetivos adicionais

De forma a combater a falha referida na concretização do principal objetivo proposto foram adicionados alguns objetivos à dissertação, descritos nos pontos seguintes. Previamente, foi descrita uma parte do trabalho anterior referente à plataforma API no âmbito de uma bolsa de investigação. Um dos objetivos dessa bolsa foi o desenvolvimento de webservices para o portal de serviços da UA e respetiva documentação no portal API. No seguimento desse trabalho foi colocado como objetivo a implementação de novos serviços web e sua documentação.

Considerando a quantidade de *webservices* já existentes no portal e tendo em conta que a tendência para implementação de mais serviços é crescente, é impensável fazer algum tipo de controlo manual das funcionalidades dos mesmos para perceber quais os serviços que se encontram funcionais ou a precisar de manutenção. Para solucionar esse problema foi pensada a utilização de uma ferramenta que permitisse ter um controlo dos *webservices* existentes com base na realização de testes unitários às funcionalidades de cada um. É importante mencionar que seriam apenas desenvolvidos testes para os serviços web implementados no âmbito desta dissertação ou da bolsa de investigação em que estive envolvida anteriormente.

Para além disso, foi adicionado como principal foco o desenvolvimento de uma solução web para controlo de perfis de acessos associados aos vários membros da Universidade de Aveiro em relação aos diversos edifícios existentes na academia. Este último ponto foi sugerido porque a atual forma de funcionamento interno do *software* de controlo de acessos não é o ideal no que respeita à atribuição de perfis de acessos aos utilizadores. Será descrito todo o processo com maior detalhe num dos capítulos seguintes.

3. Desenvolvimento de um webservice de assinaturas

3.1. Introdução

Cada membro da Universidade de Aveiro – seja docente, funcionário ou aluno – pode ser identificado de forma única e inequívoca através do chamado Utilizador Universal. Este utilizador universal está diretamente relacionado com o endereço de email de cada pessoa e, associados a este endereço de email, existem diversos dados pessoais do utilizador.

Pensou-se que seria útil o desenvolvimento de um *webservice* que tivesse em conta a informação inerente ao perfil de cada utilizador e gerasse automaticamente uma espécie de assinatura que poderia ser utilizada em diferentes contextos, por exemplo, introduzida em conteúdos HTML.

3.2. Dados dos utilizadores

Para ser possível disponibilizar parte da informação associada ao perfil de cada utilizador é necessário, em primeiro lugar, conseguir aceder à mesma de forma interna ao serviço web. Tal pode ser feito com recurso ao protocolo LDAP – *Lightweight Directory Access Protocol* – que permite a partilha de conjuntos de registos organizados em serviços de diretórios, podendo ser respetivos a diferentes tipos de elementos. **[7]**

Um dos casos mais comuns da utilização de um servidor LDAP é relativo ao fornecimento e consulta de dados de utilizadores. Esta informação é organizada de um modo hierárquico em que cada registo (designado por entrada) contém um conjunto de atributos na forma chave/valor que o caracteriza. Dos atributos possíveis de consultar podem ser indicados os mais importantes: **[8] [9]**

- **givenName**: nome próprio do utilizador;
- **sn**: abreviatura de surname, referente ao apelido do utilizador;
- **displayName**: por norma é a concatenação do nome e apelido da pessoa;
- **department**: departamento a que pertence a pessoa;
- **mail**: endereço eletrónico do utilizador;
- **memberOf**: grupos a que pertence a pessoa.

Foi com recurso a um servidor LDAP, através do acesso ao conjunto de atributos acima mencionado para cada pessoa, que foi possível fazer a manipulação da informação dos perfis dos utilizadores a ser apresentada através do *webservice*.

3.3. Armazenamento de informação

Para permitir o armazenamento dos dados necessários ao uso do *webservice* foi criada uma base de dados MySQL numa máquina virtual com o sistema operativo Windows 7 32-bit, que era acedida remotamente. Nesta máquina virtual foi instalado o XAMPP que é um ambiente de desenvolvimento para PHP: **[10] [11]**

- **X**: representa a abreviação de qualquer sistema operativo por ser multiplataforma;
- **A**pache: servidor web HTTP open-source;
- **M**ySQL: sistema de base de dados em linguagem SQL;
- **P**HP: linguagem de programação web;
- **P**erl: linguagem de programação.

Juntamente com o XAMPP foi instalado o phpMyAdmin que funciona como sistema de gestão de bases de dados MySQL e, após as instalações referidas, foi construído um script para criação da base de dados de suporte ao *webservice* de assinaturas. Este script pode ser consultado no `Anexo D – Script de criação da BD do webservice de assinaturas` e dá origem a uma base de dados denominada por db_perfis que contem na sua estrutura duas tabelas.

A tabela `utilizador` guarda os dados referentes a todos os utilizadores que já geraram tokens para assinaturas, da forma que podemos ver na `Imagem 5`.

	#	Nome	Tipo
☐	1	id_utilizador	int(11)
☐	2	email	varchar(50)
☐	3	acesso	tinyint(1)

Imagem 5 – db_perfis: estrutura da tabela "utilizador"

- **id_utilizador**: valor único incremental que representa um utilizador na BD;
- **email**: endereço eletrónico associado ao utilizador universal da UA;
- **acesso**: valor booleano que indica se o utilizador permitiu o uso dos seus dados de perfil em assinaturas.

A tabela `token` guarda os dados referentes a todos os tokens associados a assinaturas e gerados pelos utilizadores, tal como vemos na `Imagem 6`.

	#	Nome	Tipo
☐	1	id_token	int(11)
☐	2	fk_utilizador	int(11)
☐	3	token	varchar(20)
☐	4	api	int(5)
☐	5	activo	tinyint(1)
☐	6	visualizacoes	int(11)
☐	7	descricao	longtext

Imagem 6 – db_perfis: estrutura da tabela "token"

- **id_token**: valor único incremental que representa um *token* na BD;
- **fk_utilizador**: identifica quem gerou o *token* e é uma referência para o atributo id_utilizador da tabela utilizador;
- **token**: sequência de 20 caracteres gerados aleatoriamente para representar uma assinatura;
- **api**: valor constante que representa a API em que a assinatura está a ser usada, nesta fase todos os *tokens* são gerados para um único valor de API;
- **activo**: valor booleano que indica se o *token* se encontra ou não ativo;
- **visualizacoes**: número de pedidos realizados ao *webservice* para o referido *token*;
- **descricao**: pequeno texto descritivo da finalidade da assinatura.

3.4. Funcionalidades do webservice

Para permitir o desenvolvimento do *webservice* foi utilizada uma máquina virtual com o sistema operativo Ubuntu Server 14.04 64-bit, que era acedida remotamente. Esta máquina virtual já tinha sido utilizada como ambiente de desenvolvimento de *webservices* no âmbito da bolsa de investigação, daí que a configuração desse ambiente não seja parte integrante do presente trabalho.

Após a escrita do código necessário ao bom funcionamento do serviço web, ao aceder pelo *browser* ao endereço IP da máquina virtual configurada para o desenvolvimento associado ao portal de serviços da Universidade de Aveiro, é possível ver uma listagem de serviços web semelhante à apresentada no endereço atual do referido portal, incluindo entre eles o *webservice* de Assinaturas. Em frente ao nome de cada *webservice* encontra-se o símbolo da *wiki* com ligação para a página de documentação do serviço respetivo, como é possível verificar na `Imagem 7`.

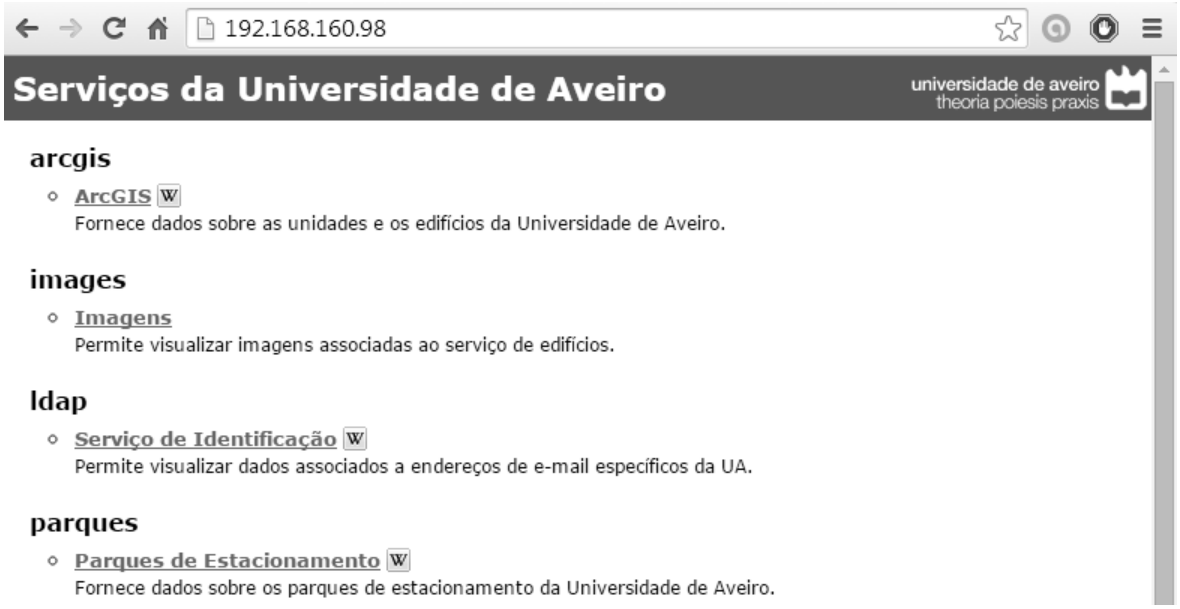

Imagem 7 – Serviços da UA: página inicial do portal de serviços da UA (ambiente de desenvolvimento)

Seguindo a ligação associada ao nome do *webservice*, a página é redirecionada para o endereço do mesmo que indica que o parâmetro `token` se encontra em falta, da forma que se vê na `Imagem 8`. Isto acontece porque para permitir a identificação unívoca de cada assinatura estas são identificadas por um código único, designado `token`, que tem associados atributos como a indicação se o mesmo se encontra ou não ativo, o número de visualizações que já foram feitas a essa assinatura e uma pequena descrição destinada ao reconhecimento da assinatura pelo utilizador.

Imagem 8 – Serviço de assinaturas: token em falta

Se ao endereço anterior for acrescentado o parâmetro `token` com um valor válido e ativo, como por exemplo `9614ce1780951115b10c`, é apresentada a informação pública sobre o utilizador que gerou esse `token`, como é possível verificar na `Imagem 9`.

Imagem 9 – Serviço de assinaturas: token válido, formato JSON

Os dados retornados por este *webservice* encontram-se, por definição, no formato JSON. Contudo, é possível indicar que se pretende que a informação seja devolvida na forma de um trecho de HTML. Se para além do parâmetro `token` for introduzido o parâmetro `f` – que representa o formato de dados a retornar – com o valor `html`, os dados apresentados podem ser imediatamente incluídos em ficheiros referentes a páginas web, à semelhança do que se vê na `Imagem 10`.

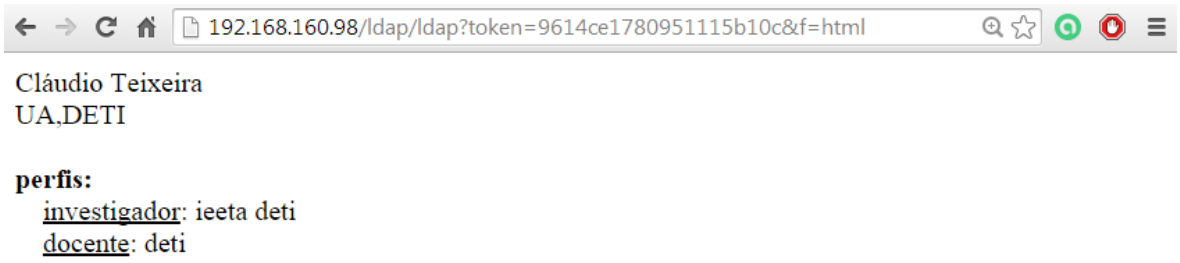

Imagem 10 – Serviço de assinaturas: token válido, formato HTML

Se for requisitada informação para um `token` que se encontra inativo, por indicação específica do utilizador que o gerou, é apresentada uma mensagem a dizer que esse `token` não se encontra disponível para consulta, ilustrada na `Imagem 11`.

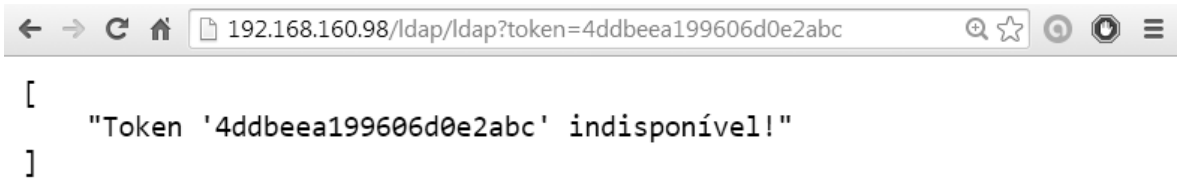

Imagem 11 – Serviço de assinaturas: token indisponível

À semelhança do caso anterior, no caso de ser requisitada informação em formato HTML para um `token` indisponível é apresentada uma mensagem que indica que a assinatura não se encontra disponível nesse momento, à semelhança do que é visto na `Imagem 12`.

Imagem 12 – Serviço de assinaturas: assinatura indisponível

Caso seja introduzido para o parâmetro `token` um valor inexistente na base de dados é mostrada uma mensagem a dizer que esse `token` é inválido, como se vê na `Imagem 13`.

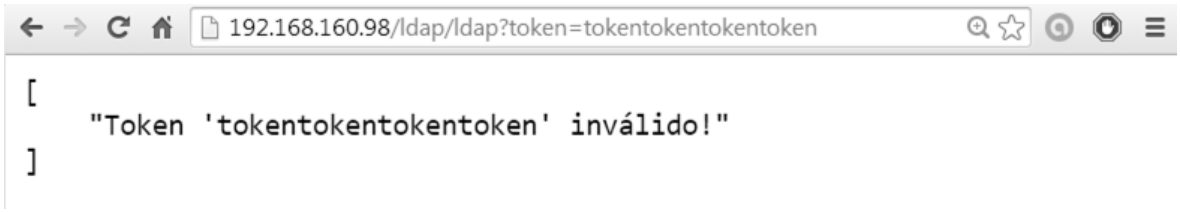

Imagem 13 – Serviço de assinaturas: token inválido

3.5. Exemplo de utilização

Para ilustrar um exemplo de utilização prático dos conteúdos deste *webservice* pode usar-se mais uma vez o `token` utilizado nos exemplos anteriores – `9614ce1780951115b10c`. Este é um `token` referente ao utilizador Cláudio Teixeira que apresenta dois tipos de perfis associados em que um deles é referente a mais de um departamento.

Como mostrado anteriormente, o aspeto dos dados apresentados em formato HTML para o referido `token` é o seguinte:

Imagem 14 – Serviço de assinaturas: informação para um utilizador específico em HTML

O grande objetivo deste *webservice* era possibilitar a geração automática de uma espécie de assinatura que poderia ser utilizada em diferentes contextos, por exemplo, introduzida em conteúdos HTML. Considerando o exemplo anterior, deveria ser possível utilizar o URL `http://192.168.160.98/ldap/ldap?token=9614ce1780951115b10c&f=html` para assinar um email construído em formato HTML. Para fazer esse teste foi escrito um email, conforme ilustra a `Imagem 15` – ainda sem assinatura.

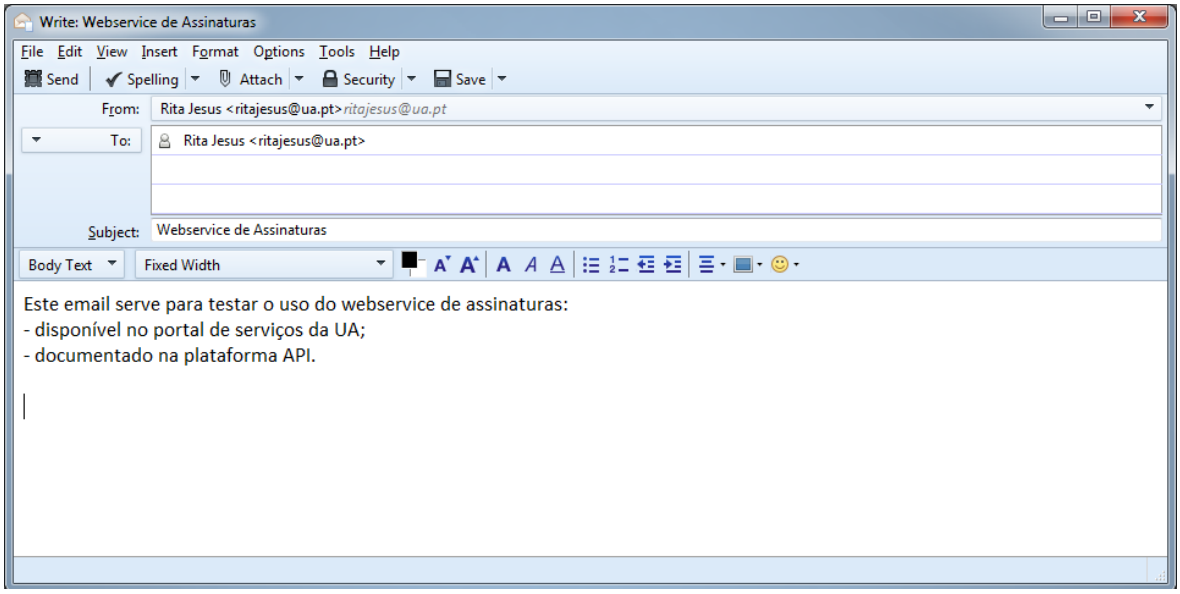

Imagem 15 – Serviço de assinaturas: novo email ainda sem assinatura

Após a redação do email de teste é necessário indicar que se pretende adicionar um trecho de código HTML – tal como se vê na `Imagem 16`.

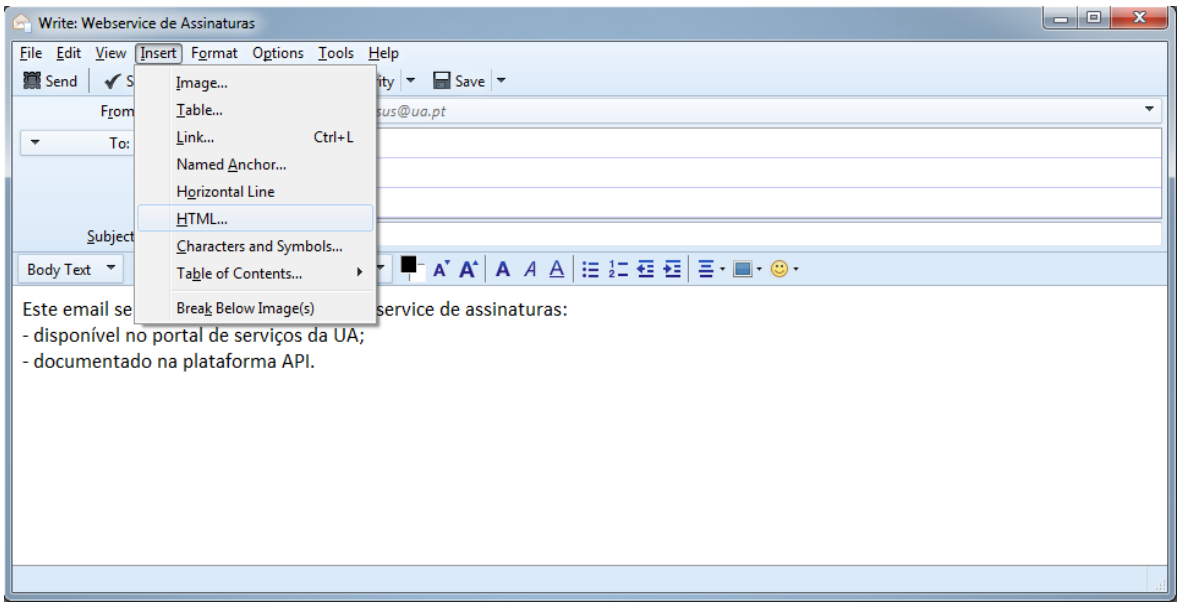

Imagem 16 – Serviço de assinaturas: escolha de introdução de código HTML no email

A Imagem 17 mostra o código necessário a introduzir para que seja mostrada a assinatura desejada no final do email.

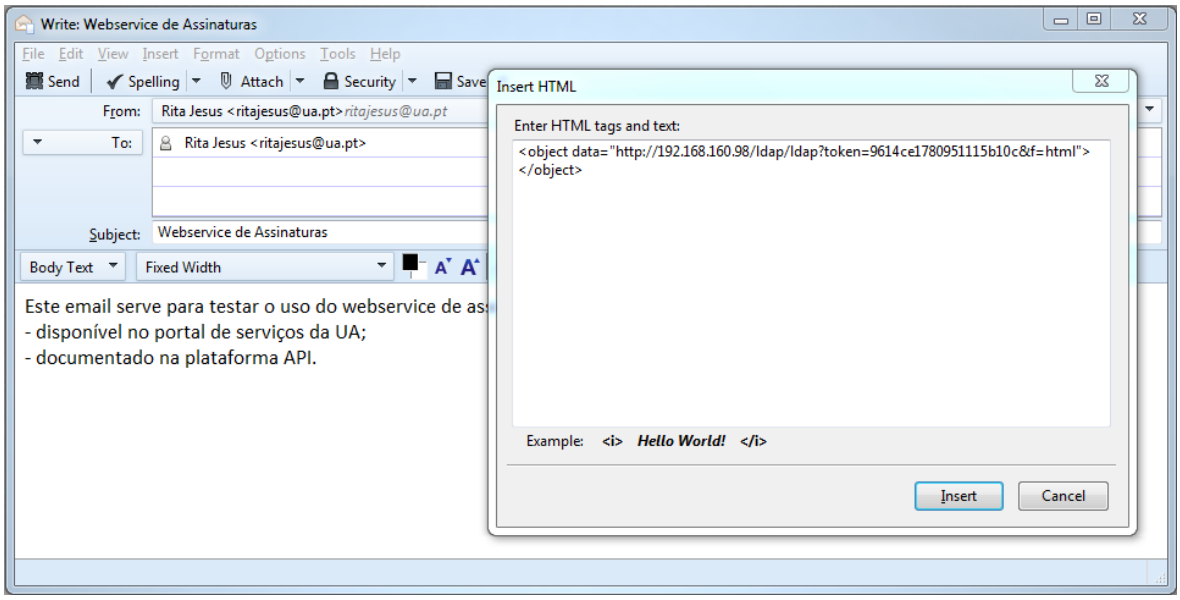

Imagem 17 – Serviço de assinaturas: trecho de código HTML introduzido para mostrar a assinatura

Depois de introduzido o pedaço de código que gera a assinatura no email, o mesmo email fica com o aspeto visível na Imagem 18.

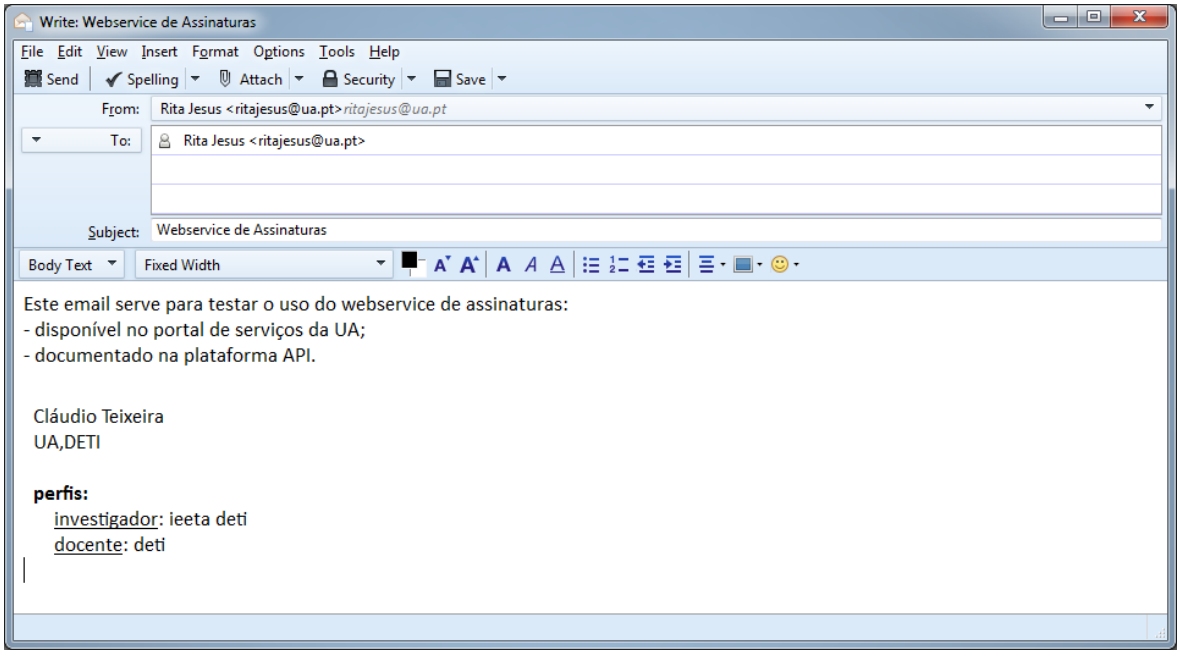

Imagem 18 – Serviço de assinaturas: novo email após introdução da assinatura

Assim que foi introduzida a assinatura no email o mesmo foi enviado, sendo recebido com o aspeto apresentado na Imagem 19.

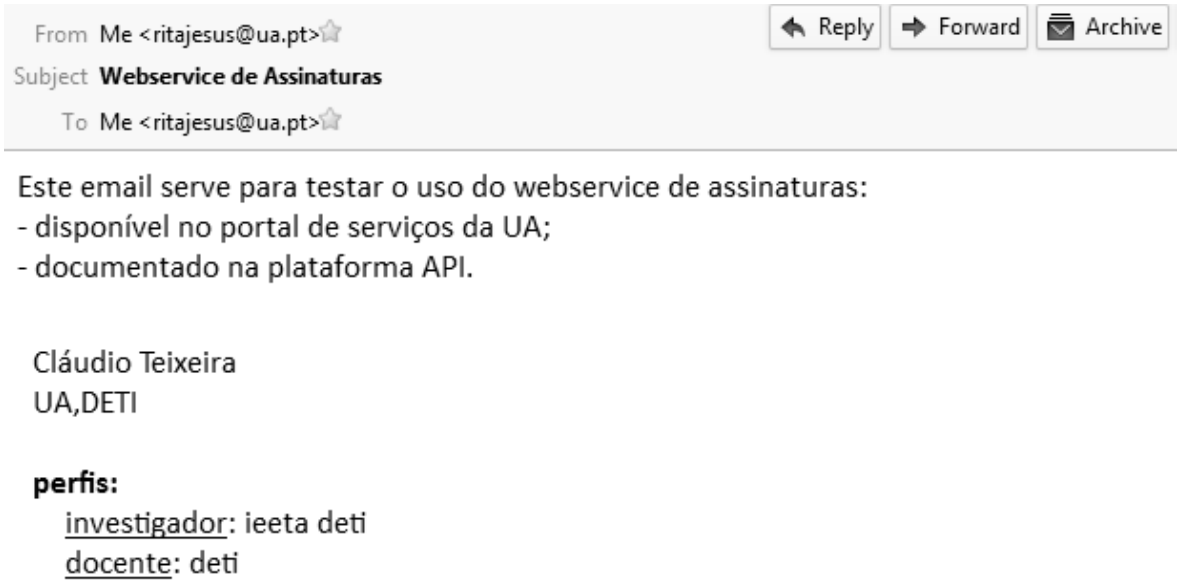

Imagem 19 – Serviço de assinaturas: aspeto do email assinado após enviado e recebido

O email do exemplo apresentado acima foi redigido e enviado para confirmar na base de dados se o número de visualizações da referida assinatura era incrementado de cada vez que o email era aberto. Este teste foi bem-sucedido uma vez que era aumentada uma visualização no registo da base de dados referente ao *token* da assinatura presente no email sempre que o mesmo era visualizado.

3.6. Documentação do webservice

É um ponto assente que o fornecimento de conteúdos através de *webservices* é a base principal de funcionamento do portal *Academic Playground & Innovation*. Contudo, para proporcionar uma boa utilização desses serviços web é necessária a existência de uma boa documentação. Este facto possibilita uma maior rapidez por parte dos utilizadores do portal na sua tomada de conhecimento das várias funcionalidades permitidas, bem como o modo mais correto para a sua utilização. Assim sendo, após o desenvolvimento do *webservice* de assinaturas, este foi também documentado de acordo com os padrões inerentes à plataforma API. Este processo pode ser consultado no `Anexo E – Documentação do webservice de assinaturas`.

4. Implementação de testes unitários

4.1. Introdução

Ao longo de todo o processo de desenvolvimento de *software* é importante garantir que a alteração de uma funcionalidade, elemento ou estrutura existente não interfere com o bom funcionamento das restantes funcionalidades, principalmente quando existem algumas com dependências entre si. Assim sendo, é importante a implementação de medidas que permitam detetar inconsistências ou incompatibilidades resultantes de alterações, podendo para isso ser construídos testes de diferentes tipos.

Um dos tipos de testes existentes são os testes unitários cujo objetivo principal passa por garantir que as funções e métodos implementados estão a funcionar da forma esperada. Cada teste unitário foca-se num pedaço de código isolado que habitualmente corresponde a um único método ou função desenvolvidos. Este processo de implementação de testes unitários pode ser aplicado sobre *webservices*. **[12] [13]**

4.2. Conceito de integração contínua

Atualmente o processo de desenvolvimento de *software* está maioritariamente associado à produção de código por diversos membros de uma equipa de trabalho, o que envolve uma boa quantidade de ficheiros que precisam de ser orquestrados em conjunto para ser possível ter um produto funcional. Este tipo de projetos costumam ter sempre associados repositórios, que permitem fazer o controlo de alterações e versões do projeto em questão, bem como quais os membros envolvidos no processo. Assim sendo, torna-se necessário dar uso ao conceito de integração contínua que permite juntar todo o código gerado pelos vários membros num único projeto e verificar a sua integridade. **[14] [15]**

A integração contínua foi originalmente pensada para uso em conjunto com a construção de testes unitários automatizados sobre o código de um projeto em ambiente local, que possibilitasse verificar se todos os testes estavam a ser realizados com sucesso antes de o novo código ser submetido. Tal facto permitia saber se o código desenvolvido por uma pessoa iria afetar o trabalho desenvolvido pelos restantes membros da equipa. Passado algum tempo do aparecimento deste conceito passaram a ser utilizados servidores que corriam os conjuntos de testes desenvolvidos de forma periódica ou automaticamente após a adição de novo código a um projeto. **[14] [16]**

4.3. Comparação de ferramentas de teste

Com a constante evolução das tecnologias ligadas aos sistemas de informação, é comum encontrar diversas ferramentas que podem ser úteis para o desenvolvimento pretendido, cada uma delas com características diferentes. Assim sendo, é muito importante fazer uma análise prévia de entre as tecnologias existentes que se enquadram nos nossos objetivos, comparando as vantagens da sua utilização antes da escolha final.

Desta forma, antes da implementação de testes unitários sobre os serviços desenvolvidos foram analisadas algumas ferramentas de integração contínua com desenvolvimento de testes unitários integrados e possibilidade de automatização dos mesmos.

4.3.1. Hudson

[17]

O *Hudson* é uma ferramenta de integração contínua que permite monitorizar a execução de tarefas realizadas de forma repetida, sendo usada principalmente durante processos de desenvolvimento de *software*. Um dos grandes pontos positivos desta ferramenta é o facto de praticamente todas as configurações necessárias após a instalação serem realizadas através de uma interface web gráfica gerada automaticamente. **[18]**

Esta ferramenta providencia um sistema de integração contínua que aumenta de forma significativa a produtividade nos projetos e reduz o tempo de procura da origem de colisões entre versões de código editadas por diferentes pessoas. Tal acontece porque quando são submetidas alterações ao código de um projeto, o *Hudson* faz a verificação das mesmas de forma automática mostrando depois o sucesso ou insucesso dos vários testes, assim como a origem de todas as falhas existentes. **[18] [19]**

É possível com o *Hudson* configurar *builds* automáticas do projeto, agendando-as para um determinado momento ou de forma periódica. Essas *builds* incluem a execução do conjunto de testes existentes para o projeto em questão, gerando relatórios que permitem verificar o estado atual do sistema e a existência de conflitos. Podem estar associadas notificações a estas *builds*, que permitem o envio de emails para um ou mais endereços específicos de forma a alertar de forma mais rápida em caso de falha. **[20]**

O *Hudson* é uma ferramenta conhecida por ser simultaneamente fácil de instalar e de fácil adaptação ao funcionamento em diversos ambientes uma vez que é multiplataforma e se encontra disponível para Windows, Linux e Mac OS. Outra das suas vantagens é o facto de permitir a sua expansibilidade através da adição de *plugins*, existindo um grande leque de *plugins* desenvolvidos para integração com esta ferramenta de forma a disponibilizar ainda mais funcionalidades. **[20]**

4.3.2. Jenkins

Jenkins **[21]**

O *Jenkins* é uma ferramenta de integração contínua muito semelhante ao *Hudson*, uma vez que também permite a monitorização de processos de desenvolvimento de *software* e de tarefas realizadas de forma repetida. À semelhança do *Hudson* também possibilita a alteração de todas as configurações através de uma interface gráfica web depois de ser feita a sua instalação e é multiplataforma, estando disponível para instalação em sistemas operativos Windows, Linux e Mac OS. **[22] [23]**

A sua utilização permite manter controlo das alterações realizadas ao código dos projetos a si associados e das *builds* relativas aos mesmos, garantindo que não ocorrem falhas não detetadas devidas à edição de código por parte de um dos membros pertencente à equipa de trabalho desse projeto. As *builds* dos projetos podem ter associadas notificações por email para que os conflitos detetados sejam corrigidos mais rapidamente. **[22] [23]**

Após alguma pesquisa para comparação entre *Hudson* e *Jenkins* foi possível entender que o motivo que leva à existência de tantas semelhanças entre estas duas ferramentas de integração contínua é o facto de o *Jenkins* ser uma *fork* do *Hudson*, isto é, o *Jenkins* foi gerado com base nas funcionalidades do *Hudson*, vindo depois a ser desenvolvido de forma independente. Assim sendo é possível enumerar algumas vantagens da utilização do *Jenkins* relativamente ao *Hudson*: **[24] [25] [26]**

- **Boa base de conhecimento**: parte da equipa que trabalhava no desenvolvimento do *Hudson* integra o desenvolvimento do *Jenkins*, incluindo o criador original, o que permite manter uma boa continuidade das funcionalidades já existentes com todo o conhecimento específico já adquirido;

- **Maior estabilidade**: desde a criação do projeto *Jenkins* foram resolvidos centenas de pequenos problemas reportados na altura em que existia apenas a ferramenta *Hudson*;
- **Mais plugins**: apesar de ambas as plataformas permitirem a inclusão de *plugins* para complemento de funcionalidades, existem mais utilizadores a desenvolver *plugins* para o *Jenkins*;
- **Fácil migração**: é possível migrar projetos *Hudson* para *Jenkins* sem perder o seu correto funcionamento;
- **Comunidade mais ativa**: a comunidade de apoio ao desenvolvimento tem estado mais ativa no *Jenkins*, sendo fornecido um maior suporte aos utilizadores nesta mesma plataforma.

Fazendo uma comparação entre as duas ferramentas descritas anteriormente, o *Jenkins* parece ser uma melhor escolha em relação ao *Hudson* uma vez que se apresenta como uma versão melhorada do mesmo e com uma maior existência de recursos e suporte relacionados com o *Jenkins*.

4.3.3. CruiseControl

[27]

O *CruiseControl* é simultaneamente uma ferramenta de integração contínua e de gestão de processos de *build* durante o desenvolvimento de *software*. Com esta ferramenta é possível garantir a monitorização das submissões de código realizadas ao longo do tempo de desenvolvimento e permite verificar se uma nova submissão causou danos ou falhas no funcionamento interno do projeto. **[28] [29]**

A sua utilização permite definir um ciclo de construção de um projeto, indicando quando é necessário desenvolver uma nova *build* do mesmo – pode ser uma construção periódica, uma construção automática devido à alteração de ficheiros fonte ou ainda uma construção pedida especificamente por um utilizador. **[28] [29]**

Esta ferramenta pode ser considerada como extensível uma vez que existem disponíveis *plugins* que podem ser integrados no contexto dos projetos associados, bastando para isso que a sua configuração seja feita no momento da sua utilização. **[28]**

À semelhança das ferramentas anteriores, esta é também multiplataforma, podendo ser utilizada e configurada em diferentes sistemas operativos. É de referir que o *CruiseControl* também possibilita o envio de emails para vários utilizadores com alertas em caso de falhas nas *builds*. **[29]**

A grande diferença entre o *CruiseControl* e as duas ferramentas descritas previamente não se prende nas funcionalidades mas sim no processo de configuração dos projetos. Esta configuração não é feita através de interfaces gráficas web, sendo necessário recorrer à edição de ficheiros em formato XML, tanto para a configuração dos projetos em si como para a adição de novas funcionalidades aos projetos. O uso deste tipo de configuração, apesar de apresentar um maior grau de complexidade, permite sobretudo uma melhor pré-configuração dos projetos e de qualquer *plugin* que se queira instalar para uso. **[30]**

4.3.4. Apache Continuum

[31]

O *Apache Continuum* é uma ferramenta de integração contínua que permite a construção de *builds* de projetos de forma automatizada, fazendo também um controlo das versões que vão surgindo. Este facto possibilita o aumento da qualidade de gestão de projetos uma vez que os membros da equipa têm facilitado o processo de gestão *builds*. Podem existir *builds* a ser realizadas de forma periódica que fazem notificações por email no caso de existirem falhas encontradas pela realização de testes. **[32] [33]**

Esta é uma ferramenta multiplataforma, disponível para os diversos sistemas operativos, que previne a quebra não notada do funcionamento de funcionalidades de um projeto, existindo forma de os vários membros do mesmo saberem que essa falha aconteceu e qual a origem do problema. **[34] [35]**

Pode dizer-se que o processo de configuração de novos projetos é semelhante ao método usado pelo *Hudson* e pelo *Jenkins* no que toca à existência de uma interface web para alteração de todas as configurações necessárias. **[36]** Contudo, o *Continuum* aparenta ter uma interface gráfica menos intuitiva e menos organizada que as duas outras ferramentas em questão, bem como uma maior dificuldade no processo de configuração e manipulação de dependências entre diferentes projetos. **[37] [38]**

4.3.5. Escolha da ferramenta de testes

Após a análise individual de várias ferramentas de integração contínua podemos verificar a presença de características comuns a todas:

- Possibilidade de desenvolvimento de testes unitários automatizados;
- Monitorização de projetos através do uso dos referidos testes;
- Agendamento de *builds* periódicas e automáticas dos projetos;
- Envio de notificações por email em caso de falha numa *build* de um projeto;
- Adaptação a diferentes sistemas operativos – ferramentas multiplataforma;
- Expansibilidade através da instalação de *plugins*;
- Melhoria da integração de código em projetos desenvolvidos em equipa;

A utilização das diferentes plataformas é de certa forma bastante semelhante em todas elas visto que a gestão dos projetos e respetivas configurações são feitas através de uma interface gráfica web. De todas as ferramentas analisadas, a única que não se enquadra neste panorama é o *CruiseControl* cujas configurações de projetos e suas propriedades são efetuadas através da edição manual de ficheiros XML.

Uma vez que já tinha abordado o *Hudson* e *Jenkins* em contexto de uma unidade curricular da Licenciatura em Tecnologias e Sistemas de Informação designada “Teste e Qualidade de Software”, ambas ferramentas que fazem uso da dita interface gráfica para configurar os projetos, e considerando que o *Apache Continuum* funciona da mesma forma que estas, *a* ferramenta escolhida para utilização neste trabalho foi o *CruiseControl*.

Esta escolha permite explorar a utilização de uma nova ferramenta e uma nova forma de configuração de projetos para monitorização através do uso de testes unitários às suas funcionalidades.

Posto isto, foi analisada uma outra ferramenta – *phpUnderControl* – que é uma aplicação adicional para o *CruiseControl* para integração contínua com projetos PHP. Esta permite novas funcionalidades ao *CruiseControl* relacionadas com o teste de *software* associado a métricas. A sua instalação é feita após a instalação do *CruiseControl*, sendo o processo de integração de ambos realizado sem necessidade de muitas configurações extra e feito de modo quase automático. **[39]**

4.4. Testes unitários desenvolvidos

Para que fosse possível a instalação do ambiente de testes utilizando a ferramenta *Cruise Control* foi criada uma máquina virtual com o sistema operativo Ubuntu Server 14.04 64-bit, que era acedida remotamente.

Uma vez que a instalação e configuração do ambiente de testes e respetivos projetos, associados aos *webservices* a testar, são processos um pouco extensos estes serão descritos em pormenor em anexos da dissertação. Assim, o processo de instalação deste ambiente de testes e das dependências necessárias para o funcionamento correto do mesmo pode ser consultado no `Anexo F – Instalação do ambiente de testes CruiseControl`. **[40] [41] [42]**

4.4.1. Webservice de parques

Foi mencionado anteriormente o desenvolvimento de *webservices* para o portal de serviços da Universidade de Aveiro no âmbito de uma Bolsa de Integração na Investigação. Um dos *webservices* desenvolvidos foi o serviço de parques que fornece dados sobre a lotação dos parques de estacionamento da UA e inclui dados como a sua capacidade, o número de lugares ocupados, o número de lugares livres e a sua localização.

A configuração inicial do projeto associado a este *webservice* no *CruiseControl* está disponível no `Anexo G – Configuração do CruiseControl para o serviço de parques`. Após essa configuração, foram implementadas funções com os seguintes objetivos:

- Testar e validar a conexão do projeto ao *webservice*;
- Confirmar o retorno de uma lista de objetos em formato JSON;
- Verificar a existência de um valor de *timestamp* nos elementos JSON;
- Garantir que a estrutura de dados dos elementos JSON corresponde à dos parques.

As funções indicadas na lista anterior permitiram o desenvolvimento de um conjunto de testes unitários que pretendiam validar de forma individual o correto funcionamento de todas as funcionalidades pertencentes a este serviço web, descritas no capítulo `2.2.1 – Webservice de parques:`

- Consulta da lista de todos os parques de estacionamento;
- Consulta dos dados de um determinado parque;
- Consulta dos dados referentes a vários parques especificados;
- Consulta dos dados relativos aos diversos parques para uma dada disponibilidade;
- Consulta dos dados associados ao conjunto parques/disponibilidade pedido.

Para cada funcionalidade exposta na lista anterior existe um teste unitário que faz a sua verificação através da validação individual de cada uma das condições apresentadas. Caso todas as condições sejam validadas com sucesso para a *build* respetiva é possível ver a apresentação na página inicial do projeto *CruiseControl* configurado, como se vê na Imagem 20:

Imagem 20 – ua_parques: projeto no dashboard construído com sucesso

Assim que a *build* é gerada é possível consultar os dados referentes à mesma, sabendo a quantidade de testes unitários que foram realizados com sucesso para esse projeto e qual o número da *build* em questão, à semelhança da que se vê na Imagem 21.

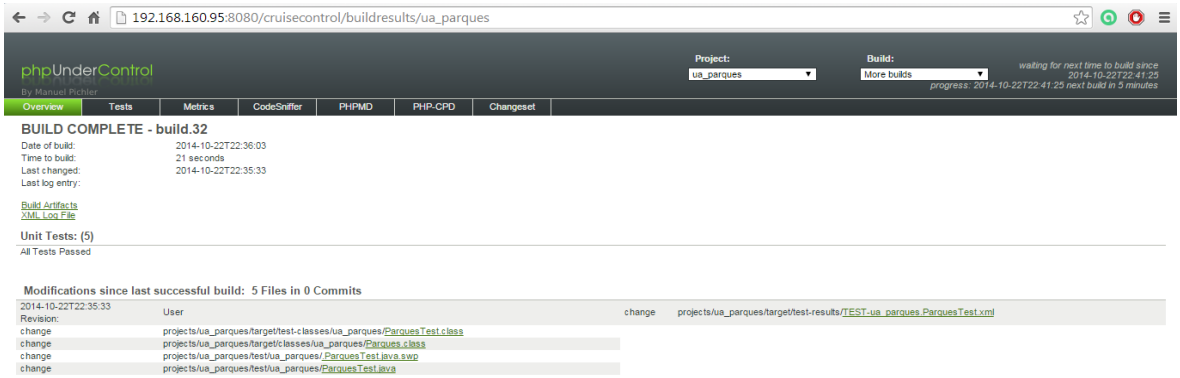

Imagem 21 – ua_parques: detalhes do projeto construído com sucesso

Caso ocorra uma falha na validação de algum dos testes unitários associados ao projeto nessa *build* é apresentado um fundo com cor alaranjada de forma a ser facilmente notado que existiu um problema na construção da *build*. É possível ver o aspeto desse cenário na Imagem 22.

Imagem 22 – ua_parques: projeto no dashboard construído com erro

Assim que a *build* é gerada para este caso é possível ver os dados da mesma como ilustrado na `Imagem 23`, sabendo quantos testes unitários existem e quais fizeram com que o processo de *build* não fosse terminado com sucesso devido à sua falha.

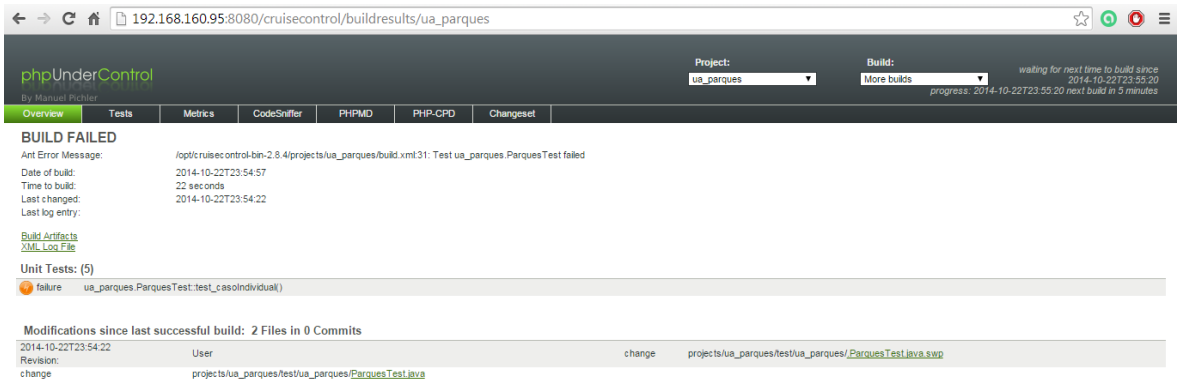

Imagem 23 – ua_parques: detalhes do projeto construído com erro

Após a implementação de todos os testes sobre o *webservice* de parques e da execução dos mesmos através de um projeto do *CruiseControl* é possível observar que todas as funcionalidades se encontram a funcionar corretamente. Contudo, continua a ser importante a existência de *builds* periódicas neste projeto para confirmar que todas as funções continuam a comportar-se de acordo com o esperado ao longo do tempo.

4.4.2. Webservice de edifícios

Para além do serviço web de parques de estacionamento da UA foi também desenvolvido no âmbito da Bolsa de Integração na Investigação o *webservice* de edifícios. Este serviço fornece dados sobre os edifícios e unidades da Universidade de Aveiro e inclui dados como coordenadas geográficas, pisos de cada edifício, infraestruturas e salas existentes em cada piso de um edifício, entre outras informações.

A configuração inicial do projeto associado a este *webservice* no *CruiseControl* está disponível no Anexo H – Configuração do CruiseControl para o serviço de edifícios. Após essa configuração do projeto, foram implementadas funções com os seguintes objetivos:

- Testar e validar a conexão do projeto ao *webservice*;
- Confirmar o retorno de uma lista de objetos em formato JSON;
- Garantir que a estrutura de dados dos elementos JSON retornados corresponde à dos edifícios.

As funções indicadas na lista anterior permitiram o desenvolvimento de um conjunto de testes unitários que pretendiam validar de forma individual o correto funcionamento de todas as funcionalidades pertencentes a este serviço web, descritas no capítulo 2.2.2 – Webservice de edifícios:

- Consulta de uma listagem com dados relativos a todos os edifícios;
- Consulta dos dados relativos aos pisos de um edifício;
- Consulta das infraestruturas de cada piso de um edifício específico;
- Consulta das informações associadas às salas de um edifício.

Para cada funcionalidade exposta na lista anterior existe um teste unitário que faz a sua verificação através da validação de cada uma das condições apresentadas e da estrutura de dados inerentes a cada tipo de dados a retornar – edifício, piso, infraestrutura ou sala. Caso todas as condições sejam validadas com sucesso para uma dada *build* é possível ver a página inicial do projeto *CruiseControl* da forma mostrada na Imagem 24.

Imagem 24 – ua_edificios: projeto no dashboard construído com sucesso

Assim que a *build* é gerada é possível consultar os dados referentes à mesma, sabendo a quantidade de testes unitários que foram realizados com sucesso para esse projeto e qual o número da *build* em questão, como se pode ver na Imagem 25.

Imagem 25 – ua_edificios: detalhes do projeto construído com sucesso

À semelhança do que aconteceu no caso do serviço web de parques, após implementação de todos os testes unitários sobre o *webservice* de edifícios e da execução dos mesmos através de um projeto do *CruiseControl* é possível observar que todas as funcionalidades se encontram a funcionar corretamente. Contudo, continua a ser importante a existência de *builds* periódicas neste projeto para confirmar que todas as funções vão continuar a comportar-se de acordo com o esperado ao longo do tempo.

5. Componente para gestão de perfis de acesso

5.1. Introdução

É possível encontrar o conceito de controlo de acessos associado aos mais diversos tipos de objetivos, sendo os mais importantes o controlo de acessos a nível de elementos físicos ou a nível de segurança de informação. Para isso é necessária a existência de permissões de acesso que identificam as autorizações para os recursos em questão. **[43]**

O exemplo mais comum para segurança física de edifícios é o controlo de abertura e fecho de portas utilizando fechaduras e chaves, sendo muitas vezes o único fator de segurança para entrada em edifícios, como por exemplo em casas de habitação pessoal. Em algumas moradias, habitualmente situadas em zonas de moradores com maiores capacidades financeiras, existe a presença de um segurança na entrada principal que faz algum controlo relativamente aos acessos ao edifício em questão.

Contudo, para empresas ou outras organizações de grande dimensão, a existência destes dois processos de segurança descritos anteriormente apresenta fragilidades e pode não ser o suficiente. Para além de ser necessário saber se uma determinada pessoa tem ou não permissões para passar da entrada de um edifício, às vezes é também importante saber quais as zonas específicas desse edifício a que a pessoa pode aceder.

Para resolver esse problema podem ser utilizados meios tecnológicos, que podem passar por mecanismos de controlo de acessos associados a elementos físicos – como cartões ou outro tipo de dispositivo móvel – que identificam de forma exata quem está a aceder ao local e em que momento.

5.2. Controlo de acessos na Universidade de Aveiro

Tendo em conta que a Universidade de Aveiro é uma instituição de ensino, é fácil saber que em cada um dos seus edifícios existem conteúdos com valores muito elevados, tanto a nível de materiais como de informação, sendo a segurança destes edifícios um fator de extrema importância para garantir que nada acontece aos mesmos.

Desta forma, em todos os edifícios existem dispositivos que permitem registar e controlar todos os acessos através de cartões associados ao Utilizador Universal de cada membro pertencente à academia – seja docente, funcionário ou aluno.

Para que seja possível uma boa gestão dos acessos realizados por cada pessoa em cada edifício, cada utilizador tem associado um conjunto de permissões e são serviços de Tecnologia de Informação e Comunicação (*sTIC*) da Universidade de Aveiro que estão responsáveis pelo *software* associado a este controlo de permissões e acessos.

Após alguma análise, os *sTIC* concluíram que o modo atual de funcionamento interno para atribuição de permissões e acessos aos utilizadores não é o ideal. Neste momento existe a possibilidade de vários utilizadores estarem associados ao mesmo perfil de acesso, facto que dificulta o processo de atualização de perfis porque para acrescentar ou remover acessos de um perfil para um único utilizador tem de ser criado um novo perfil quase idêntico.

Considerando assim a necessidade de reestruturar e simplificar o processo de atribuição de acessos aos utilizadores e, uma vez que a dinamização da plataforma API não pôde seguir o rumo com o alcance pretendido, este ponto tornou-se um dos principais focos da dissertação. O objetivo central deste componente será possibilitar a atribuição de um perfil único por utilizador que possa ser alterado sem gerar conflitos com outros utilizadores ou outras funcionalidades já existentes.

5.3. Instalação do sistema de controlo de acessos atual

Para que fosse possível modificar o modo de funcionamento interno de atribuição de perfis de acesso aos utilizadores era necessário entender quais os dados envolvidos no processo de associação de um perfil de acesso a um utilizador.

Desta forma, foi necessário fazer a instalação do sistema atual de controlo de acessos designado por *Net2 Access Control*. Apesar deste *software* ser licenciado da *Paxton* existe uma versão que pode ser transferida e instalada gratuitamente para fins exemplificativos, bastando selecionar a opção "*Net2 Demo*".

Antes da instalação do sistema de controlo de acessos propriamente dito foi instalado em primeiro lugar o Microsoft SQL Server 2008, uma vez que o *Net2 Access Control* tem como suporte de informação uma base de dados SQL Server.

Considerando que a solução pretendida para o processo alterado de atribuição de perfis de acesso aos utilizadores seria uma solução web, foi também instalado o Microsoft Visual Studio 2012.

Após instalado o *software* de controlo de acessos foi possível aceder imediatamente ao mesmo, bastando introduzir como credenciais de acesso ao sistema demonstrativo o nome de utilizador “Técnico do sistema” e a palavra-passe “net2” – ver `Imagem 26`.

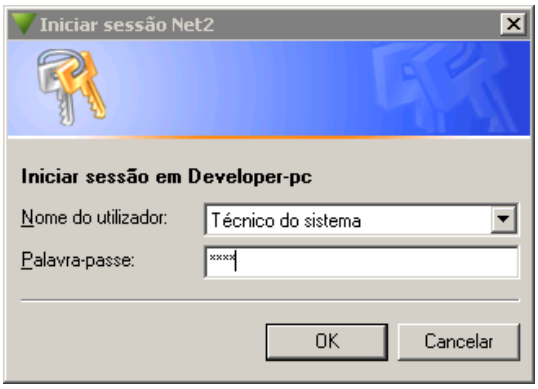

Imagem 26 – Net2: credenciais de entrada no software

Assim que é feita a entrada no sistema é apresentado o painel principal de gestão com diversos menus e funcionalidades, à semelhança do que se vê na `Imagem 27`. Os dois menus mais importantes para as alterações e desenvolvimentos que serão descritos mais à frente são os menus de Utilizadores e de Níveis de Acesso.

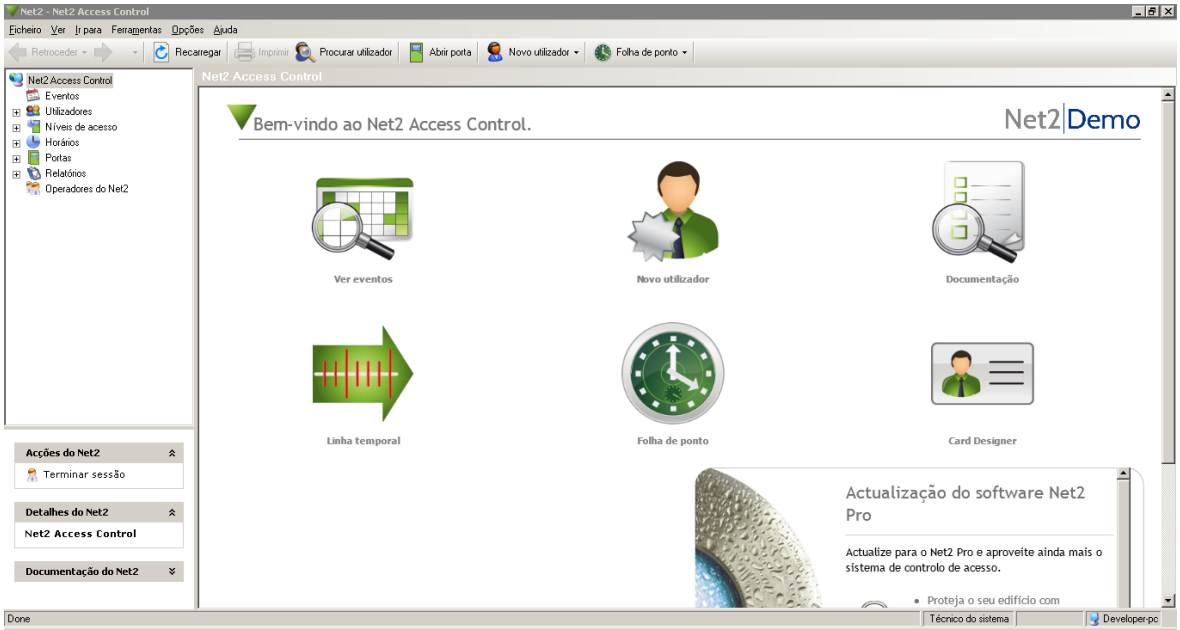

Imagem 27 – Net2: painel principal do software de controlo de acessos atual

Como é possível ver na `Imagem 28`, os utilizadores estão distribuídos por diferentes categorias e, a cada um deles, está associada alguma informação pessoal para além das autorizações de níveis de acesso.

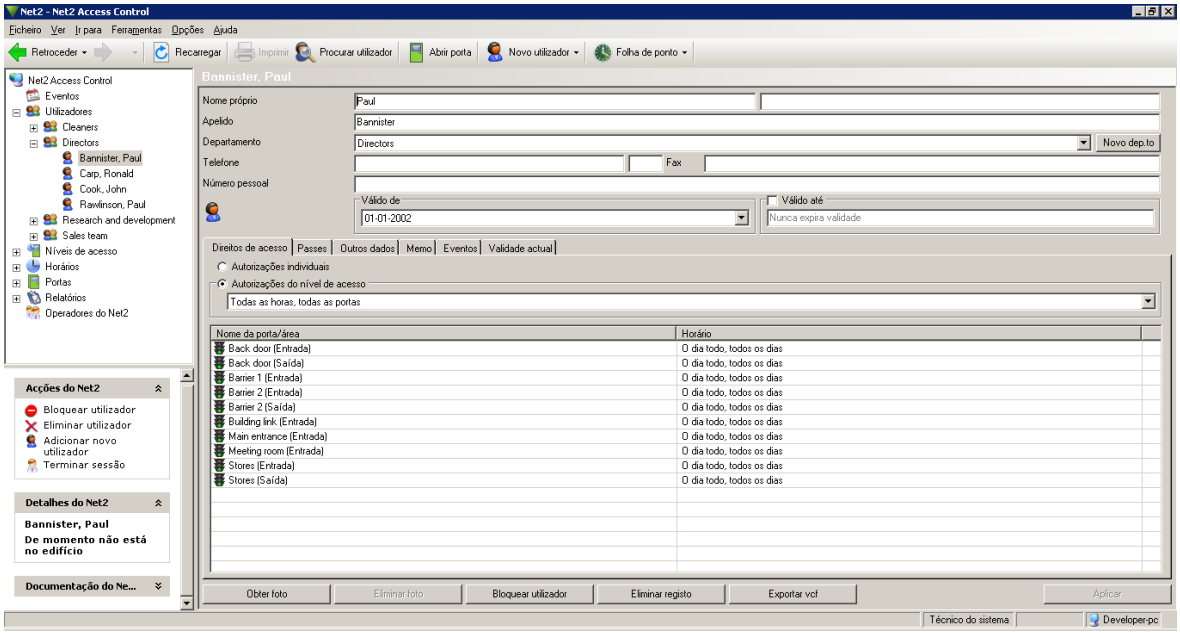

Imagem 28 – Net2: dados de um utilizador com acesso associados

Relativamente aos níveis de acesso, é possível ver para cada um deles quais as portas a que se refere e qual o horário de acesso atribuído a cada uma delas, da forma apresentada na Imagem 29.

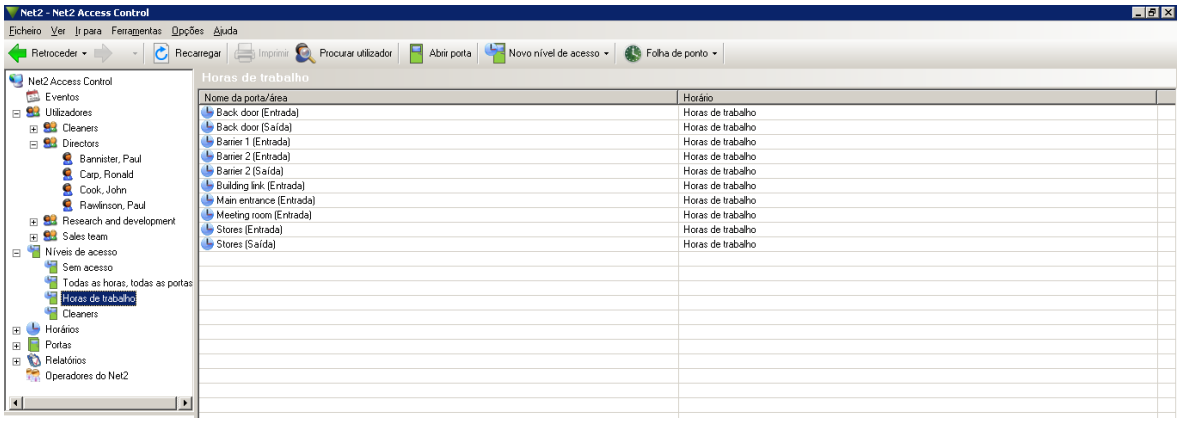

Imagem 29 – Net2: zona para controlo de acessos e seus horários

Durante a instalação deste *software* foi criado automaticamente um novo servidor de base de dados SQL Server identificado por NET2 que armazena todos os dados necessários ao seu bom funcionamento. Para que fosse possível analisar a estrutura da base de dados nesse servidor foram feitas algumas configurações que podem ser consultadas no Anexo I – Configuração do servidor de base de dados NET2.

5.4. Análise do funcionamento do sistema de controlo de acessos

Foi mencionado anteriormente que o software atual de controlo de acessos usado tem os seus dados suportados numa base de dados SQL Server. Considerando este facto, foi utilizada uma ferramenta associada ao SQL Server – designada SQL Server Profiler – que permite criar, gerir e analisar diferentes sequências de comandos numa base de dados específica.

Neste caso o SQL Server Profiler foi usado inicialmente para investigar quais os comandos enviados pelo software de controlo de acessos para a base de dados no momento da criação de um novo utilizador e respetivo perfil. Ao tentar aceder ao SQL Server Profiler só é possível avançar após escolher qual o servidor a ser conectado para análise. Depois de realizada essa conexão tem de ser configurado o ficheiro onde se pretende que sejam guardados os dados da análise para consulta posterior, da forma mostrada na Imagem 30.

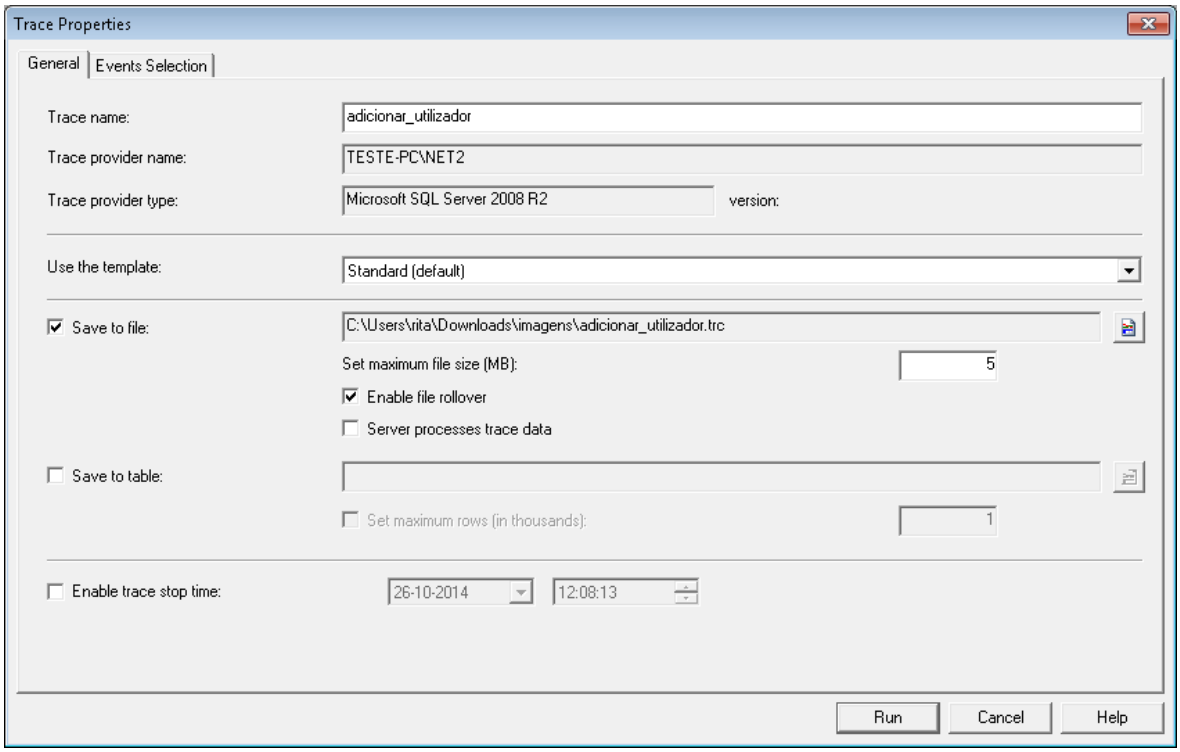

Imagem 30 – Profiler: configuração de um ficheiro para uma nova análise

Como podemos ver na Imagem 31, uma análise realizada pelo SQL Server Profiler permite a seleção de vários tipos de eventos e dados. Para que seja possível uma melhor perceção do objetivo de cada um destes tipos de dados é feita uma breve descrição de cada um destes no Anexo J – Descrição dos elementos analisados pelo SQL Server Profiler.

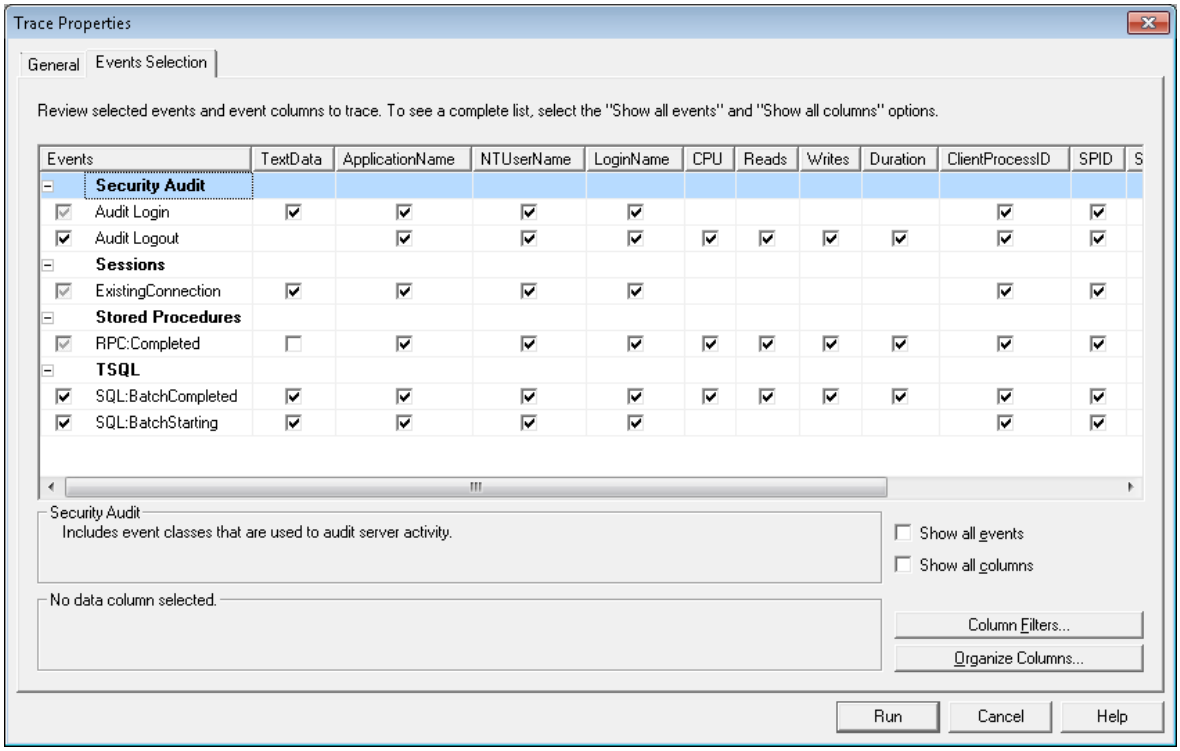

Imagem 31 – Profiler: eventos possíveis numa análise do SQL Server Profiler

Numa fase inicial as pesquisas foram realizadas com todos os parâmetros selecionados. Após ter sido iniciado o processo de captura de comandos pelo SQL Profiler, foi necessário criar um novo utilizador no *software* Net2 Access Control. Tal funcionalidade pode ser acedida diretamente através do painel principal que é mostrado ao entrar no sistema. Depois de escolhida essa funcionalidade aparece um formulário – ilustrado na `Imagem 32` – para preenchimento dos dados a associar ao novo utilizador.

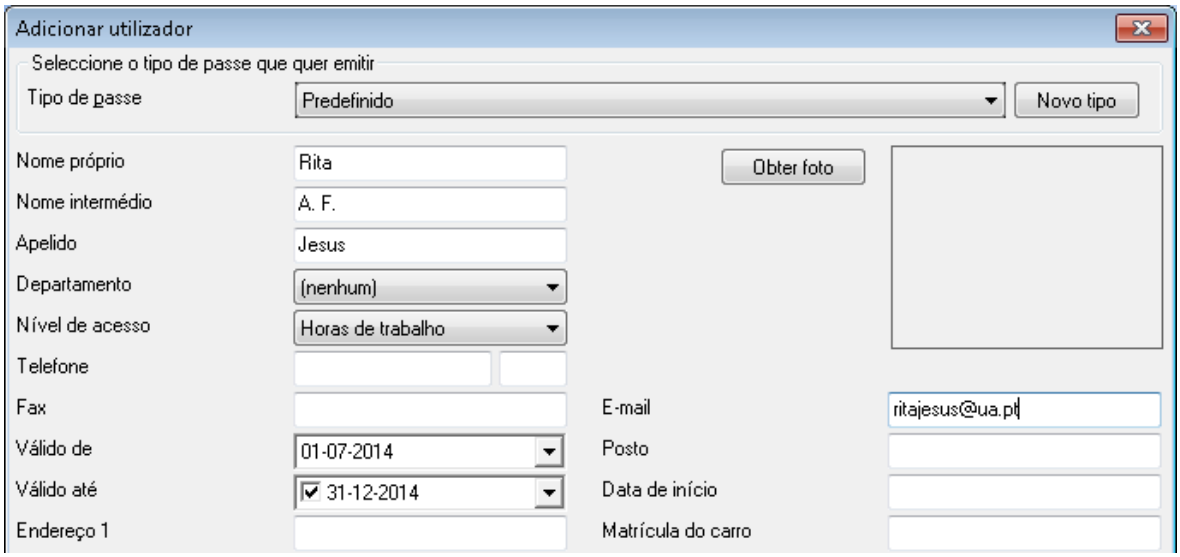

Imagem 32 – Net2: formulário para criação de um novo utilizador

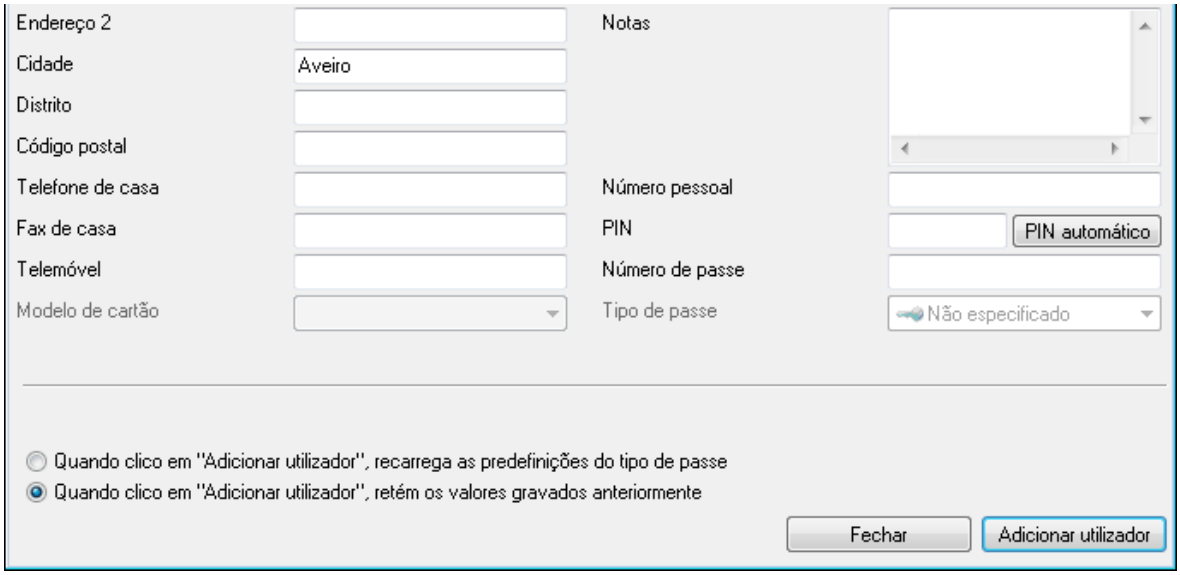

Imagem 33 – Net2: formulário para criação de um novo utilizador (continuação)

Após a introdução desses dados no formulário e depois de o utilizador ser adicionado ao sistema foi necessário voltar ao SQL Profiler para parar o processo de captura de informação, obtendo um resultado igual ao apresentado na `Imagem 34` – o conjunto de comandos executados pelo *software* para comunicação com a base de dados durante o processo.

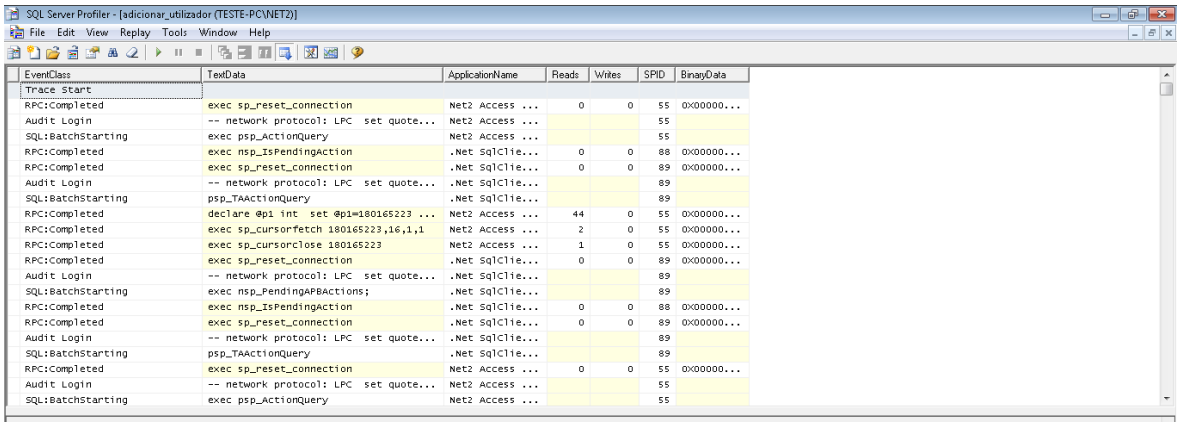

Imagem 34 – Profiler: conjunto de dados capturados pelo SQL Server Profiler

Uma vez que por cada segundo são capturadas bastantes linhas de comunicação pelo SQL Server Profiler foi necessária uma análise bastante focada e exaustiva dos mesmos até entender quais os comandos usados para criação do novo utilizador e respetivo perfil. Este foi um processo demorado porque teve de ser analisada cada linha individualmente de forma manual e sistemática para cada exemplo testado.

5.4.1. Estrutura da base de dados

Após a análise dos comandos resultantes das diversas capturas com o SQL Server Profiler conclui-se que, para além da tabela "Users" que armazena todos os dados relacionados com os membros da Universidade de Aveiro, existem quatro tabelas de grande importância no processo de gestão do controlo de acessos. Estas tabelas estão relacionadas da forma que podemos ver no modelo apresentado na `Imagem 35`.

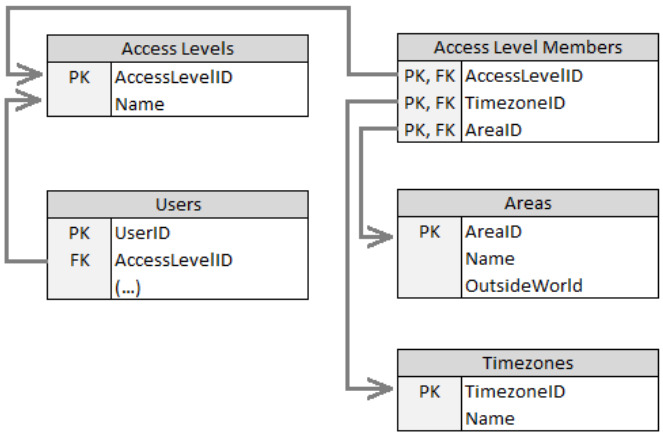

Imagem 35 – BD Net2: modelo relacional para as tabelas manipuladas

Tabela "Access Level Members"

Cada registo da tabela `Access Level Members` – ver `Imagem 36` – contêm três atributos diferentes, sendo cada um destes um identificador único respetivo a uma tabela distinta:

- AccessLevelID: identificador do nível de acesso;
- TimezoneID: identificador do horário atribuído a uma área;
- AreaID: identificador da área a aceder.

	AccessLevelID	TimezoneID	AreaID
1	0	0	2
2	0	0	3
3	0	0	12
4	0	0	13
5	0	0	22
6	0	0	23
7	0	0	32
8	0	0	33
9	0	0	42
10	0	0	43

Imagem 36 – BD Net2: tabela "Access Level Members"

Esta é a tabela mais importante a nível de controlo de acessos por ser a tabela que faz a relação entre cada permissão de acesso e as suas respetivas áreas e horários.

Tabela "Access Levels"

Esta tabela – representada na Imagem 37 – contém os identificadores dos níveis de acesso e respetivos nomes.

	AccessLevelID	Name
1	0	Sem acesso
2	1	Todas as horas, todas as portas
3	2	Horas de trabalho
4	3	Cleaners

Imagem 37 – BD Net2: tabela "Access Levels"

Tabela "Timezones"

Esta tabela – apresentada na Imagem 38 – contém os identificadores dos horários e respetivos nomes.

	TimezoneID	Name
1	0	Nunca
2	1	O dia todo, todos os dias
3	2	Horas de trabalho
4	3	Cleaners

Imagem 38 – BD Net2: tabela "Timezones"

Tabela "Areas"

Esta tabela – que pode ser vista na Imagem 39 – contém os identificadores das áreas e respetivos nomes. Para além disso ainda tem um outro atributo – *OutsideWorld* – que indica se a área é externa ou interna.

	AreaID	Name	OutsideWorld
1	2	Back door (Entrada)	0
2	3	Back door (Saída)	0
3	12	Barrier 1 (Entrada)	0
4	13	Barrier 1 (Saída)	0

Imagem 39 – BD Net2: tabela "Areas"

Considerando os dados apresentados pelas quatro tabelas anteriores podemos concluir que cada nível de acesso pode ter vários utilizadores a si associados. O que é pretendido com a nova solução é que cada perfil de acesso apenas esteja atribuído a um utilizador, o que permite que cada perfil seja identificativo. Assim sendo, foi necessário fazer algumas alterações na estrutura da tabela *Access Level Members* na base de dados do NET2 de modo a que cada um dos seus registos passasse a estar relacionado com um único utilizador. As alterações efetuadas podem ser vistas no Anexo K – Alterações à tabela "Access Level Members" do NET2.

Após alterado, o modelo relacional passou a estar estruturado da forma que podemos ver na Imagem 40. É importante mencionar que não existem visualmente alterações muito significativas neste modelo uma vez que todo o processo de manipulação de dados se encontra a ser processado através de *triggers* e *stored procedures*, descritos no Anexo K – Alterações à tabela "Access Level Members" do Net2 e Anexo L – Stored Procedures criados para a BD do Net2.

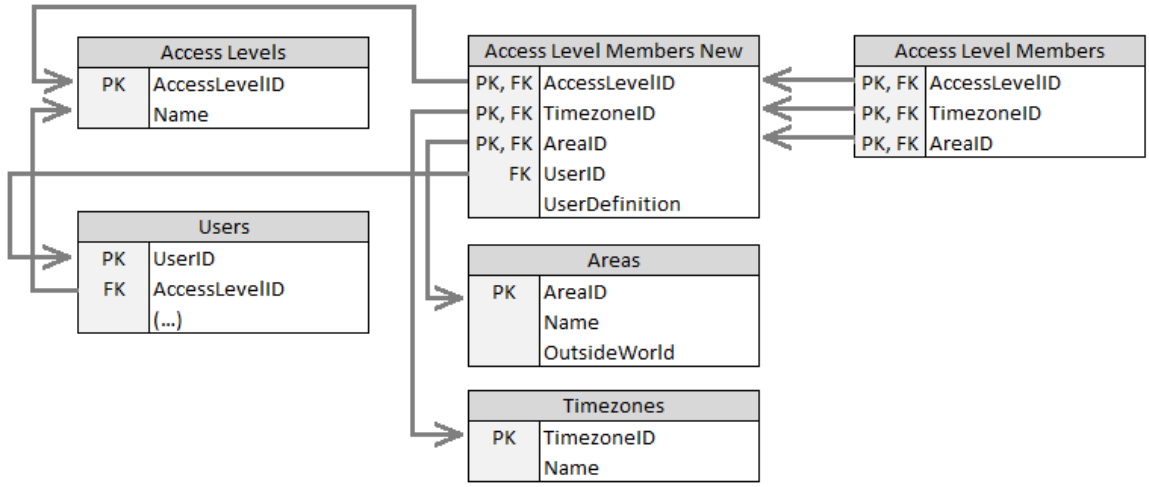

Imagem 40 – BD Net2: modelo relacional para as tabelas manipuladas após as alterações

5.5. Introdução de dados reais na base de dados

Com a finalidade de testar o *software* de controlo de acessos após as alterações com dados reais foi facultada uma cópia dos dados existentes filtrados apenas para um dos edifícios da Universidade de Aveiro – o Instituto de Telecomunicações – tendo sido disponibilizados 6 ficheiros para construção do servidor SQL de testes:

- Ficheiros mdf: SQL Server Database Primary File;
- Ficheiros ldf: SQL Server Database Transaction Log File.

Cada uma destas extensões tinha três ficheiros associados, um para cada uma das bases de dados que compõem o servidor SQL NET2 – um relativo à base de dados "Net2", outro relacionado com a base de dados "Net2 Archive" e ainda outro referente à base de dados "Net2 Events". Antes de adicionar estes ficheiros ao servidor foi necessário remover as bases de dados de demonstração existentes, uma vez que se pretende que as mesmas sejam substituídas. Após esse passo adicionaram-se então as bases de dados com dados reais ao servidor SQL NET2, através de um processo de *attach* – Imagem 41.

Imagem 41 – BD Net2: opção de attach de uma base de dados ao servidor SQL

Para cada uma das bases de dados que se pretendia associar ao servidor SQL do Net2 foi realizado o mesmo processo, selecionar o ficheiro .mdf do sistema e guardar as alterações, como se vê na `Imagem 42`.

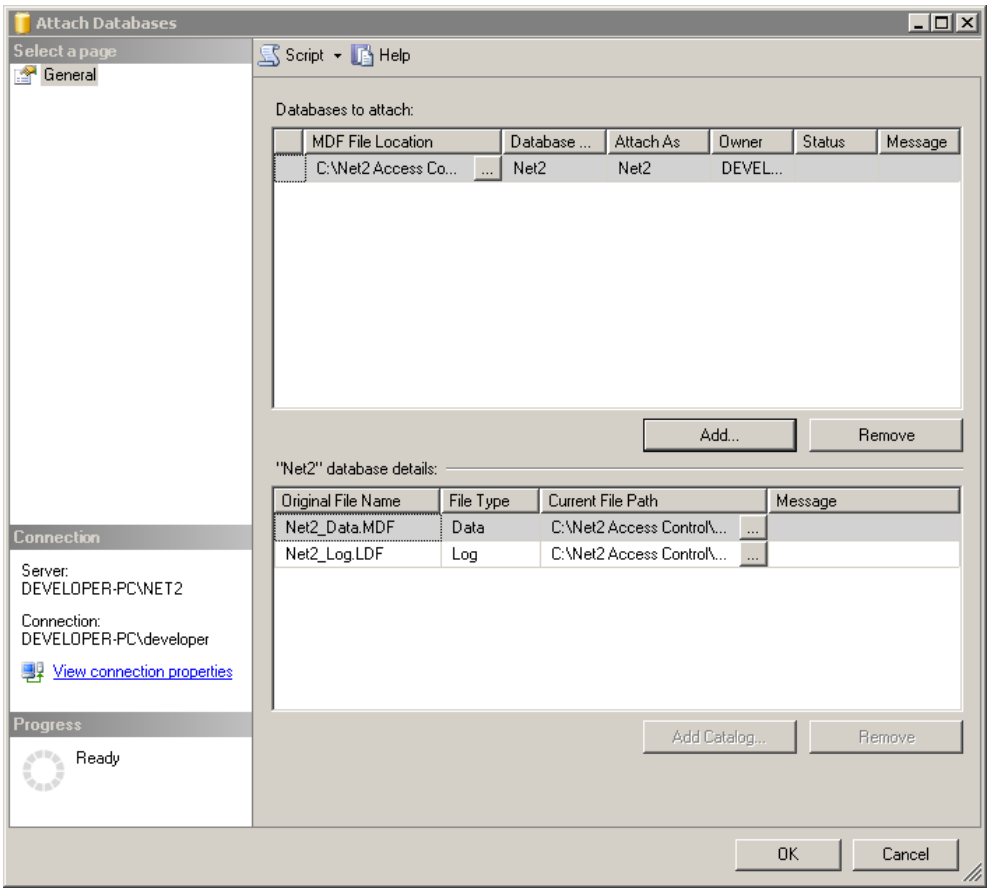

Imagem 42 – BD Net2: attach de um ficheiro .mdf para importação de uma base de dados

Após adicionadas as bases de dados ao servidor teve de ser executado novamente o script de alteração da base de dados e reiniciados os serviços SQL Server (Net2), Net2 Client Service e Net2 Service. Depois do processo descrito, ao selecionar a opção de listar todos os utilizadores através do *software* de controlo de acessos podemos ver parte dos nomes de utilizadores reais associados à base de dados de perfis de acesso na UA, referentes ao Instituto de Telecomunicações – como ilustrado na `Imagem 43`.

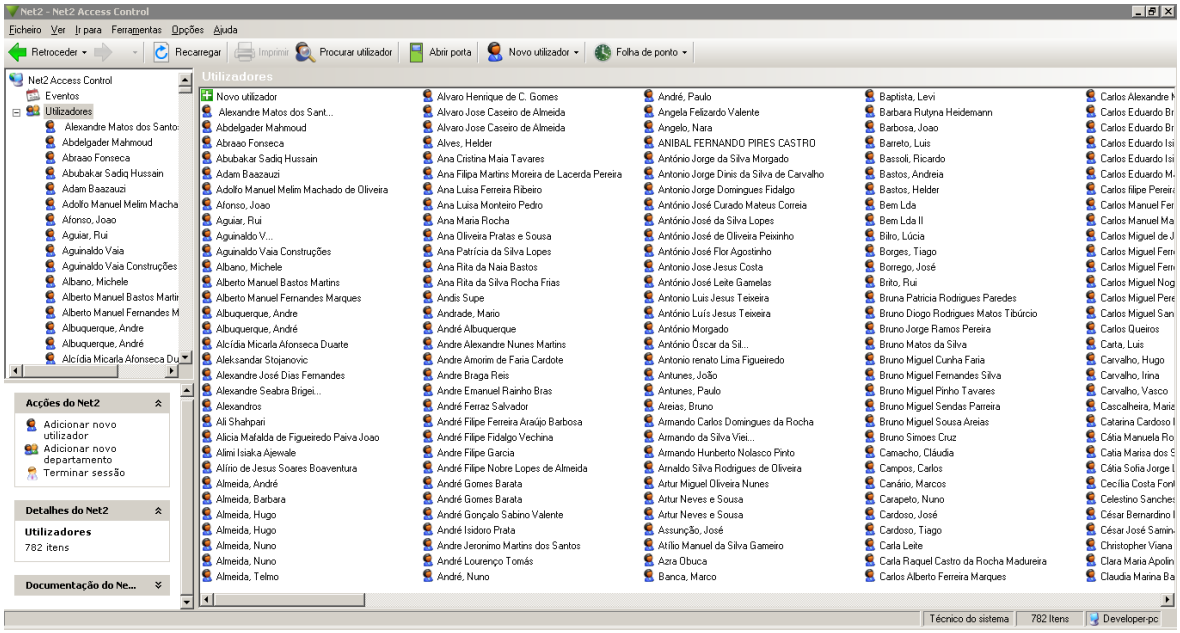

Imagem 43 – BD Net2: listagem de parte dos utilizadores do IT

Ao ser selecionado o separador de níveis de acesso é possível também ver a listagem de todos os níveis de acesso existentes para este edifício – como na `Imagem 44`.

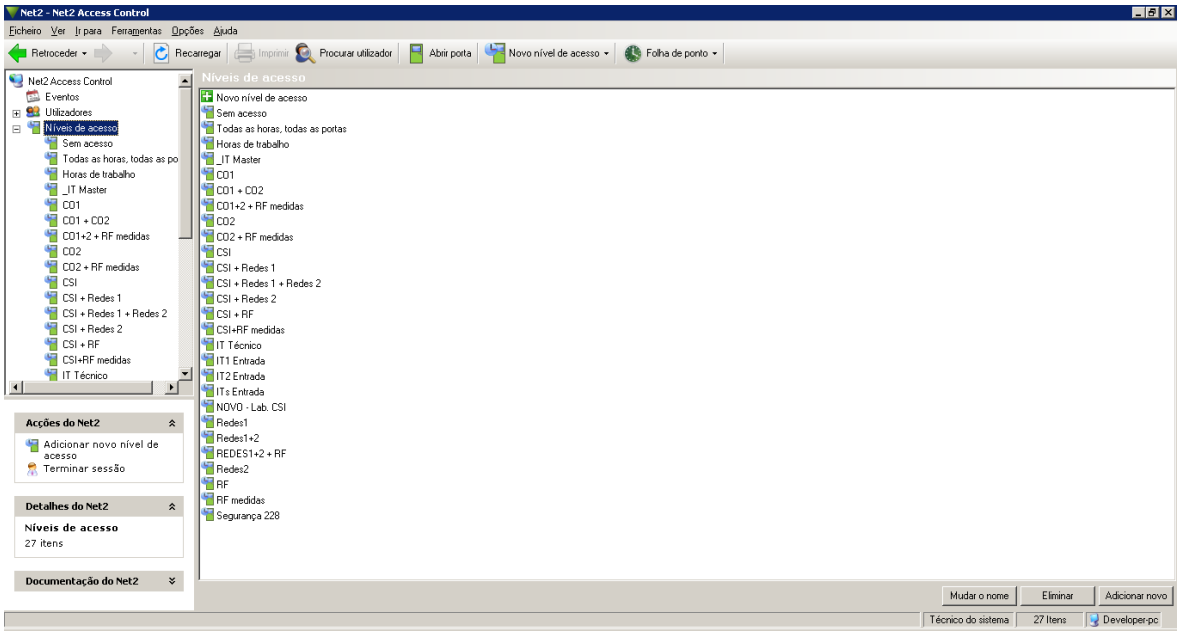

Imagem 44 – BD Net2: listagem dos níveis de acesso do IT

De igual forma, se for selecionado o separador de horários é possível ver a lista de todos os horários que podem ser associados a níveis de acesso – ver `Imagem 45`.

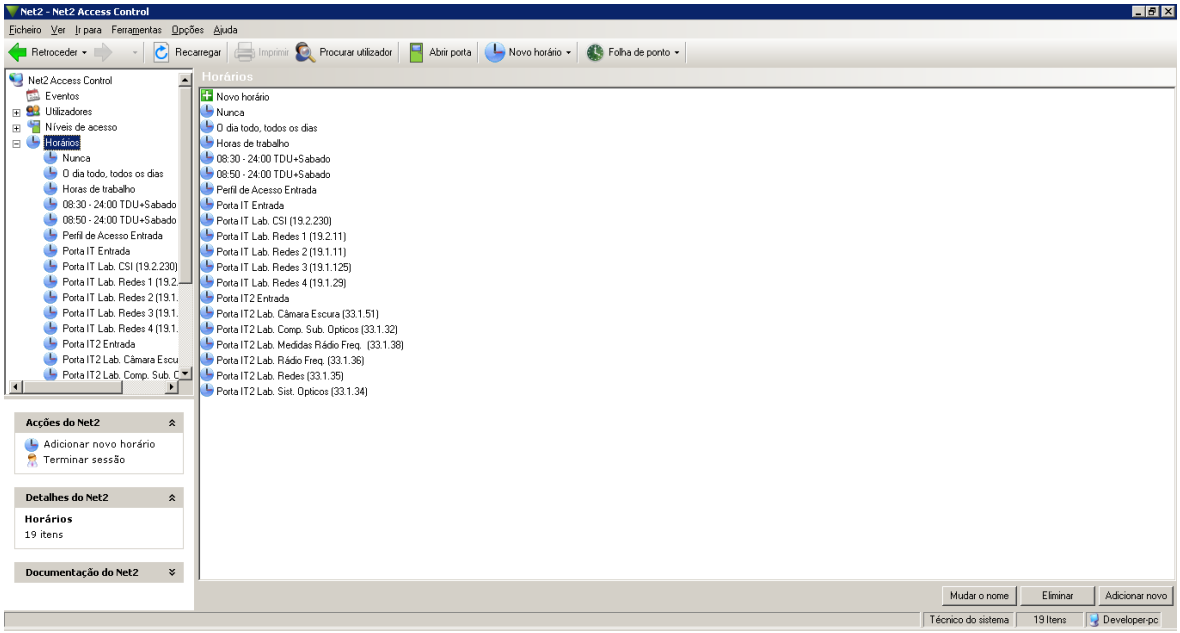

Imagem 45 – BD Net2: listagem de horários do IT

5.6. Desenvolvimento de uma interface web

As alterações feitas até ao momento permitem que os dados apresentados no *software* de controlo de acessos atual sejam de utilizadores reais com acessos atribuídos no Instituto de Telecomunicações da Universidade de Aveiro. Como referido anteriormente, o grande objetivo destas modificações visa a construção de uma interface web que permita uma melhor gestão das permissões de acesso de cada utilizador com base na atribuição de um novo tipo de perfis, que são os perfis individuais.

As questões relacionadas com a segurança da plataforma a nível de logins para visibilidade e edição dos dados não foi considerada uma necessidade no âmbito desta dissertação, uma vez que este seria um componente a desenvolver para posterior integração numa solução web existente e com toda a parte de segurança implementada.

O mesmo acontece com o *design* da plataforma, como o portal onde seria feita a integração já tem um *design* próprio esse aspeto não foi considerado como parte do desenvolvimento do componente. A `Imagem 46` apresenta uma sugestão de apresentação da página inicial associada a esta componente web que apresentaria uma lista das funcionalidades base disponíveis.

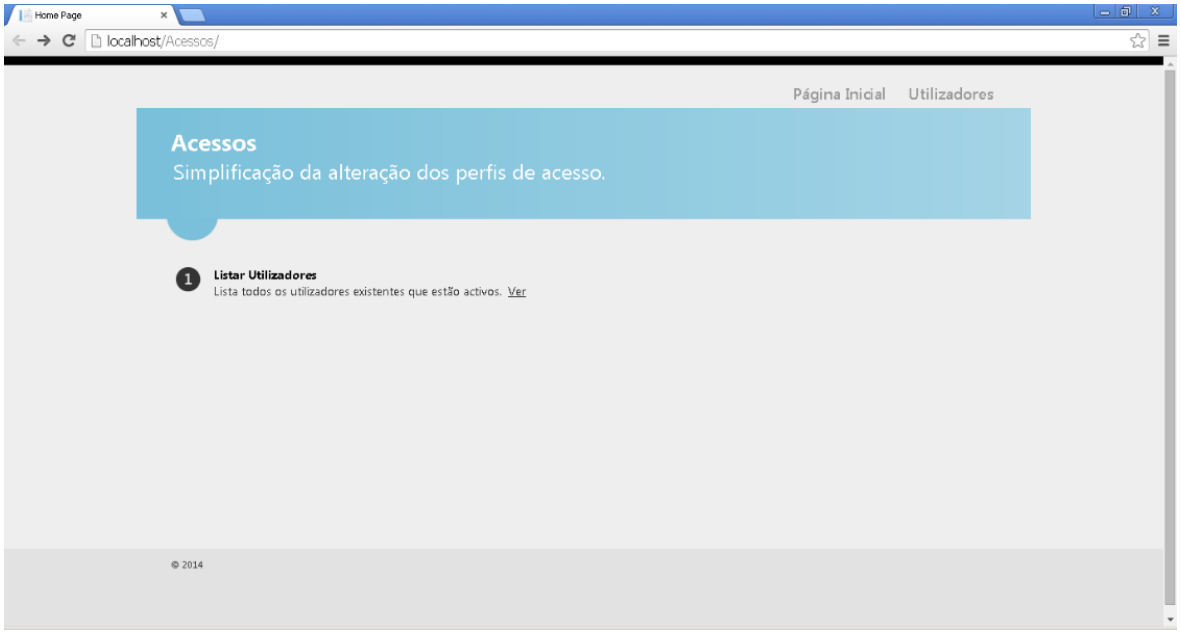

Imagem 46 – Interface web: página inicial

Selecionando a opção de ver a lista de utilizadores ativos é apresentada ao utilizador uma listagem paginada completa de todos os utilizadores existentes ordenados por data de modificação. Para cada um destes é possível ver se o perfil de acesso que tem atribuído é unicamente seu ou partilhado com outros através do campo "perfil individual?". É ainda possível selecionar a possibilidade de alterar os dados relativos ao seu perfil de acesso carregando em cima do botão de edição – como é mostrado na `Imagem 47`.

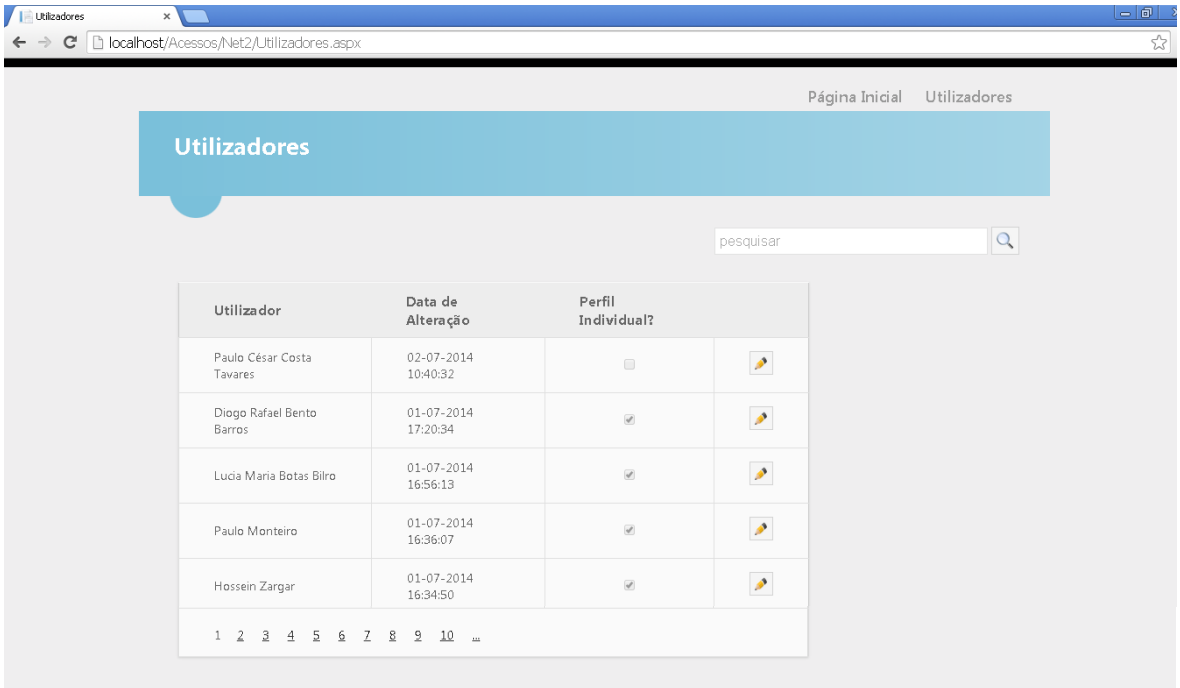

Imagem 47 – Interface web: listagem de utilizadores na plataforma

Considerando como exemplo o primeiro utilizador que aparece na lista foi selecionada a opção de edição de dados do seu perfil de acesso. É possível ver na `Imagem 48` que para esta funcionalidade é apresentada uma lista das áreas existentes no edifício bem como as suas permissões de acesso atuais para cada uma dessas áreas e uma notificação que indica se a área em questão está ativa ou inativa.

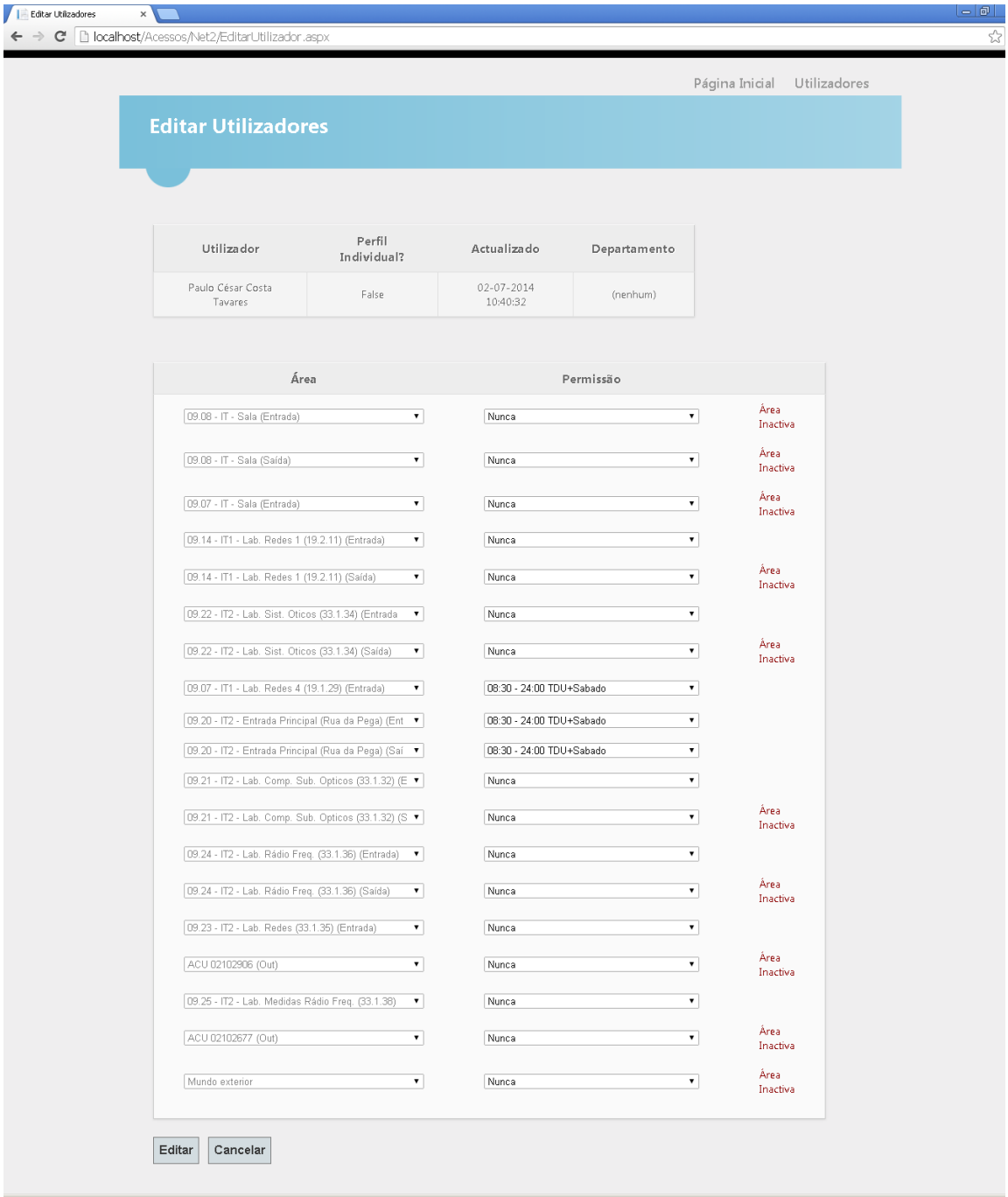

Imagem 48 – Interface web: níveis de acesso de um utilizador específico

Na página de utilizadores com perfis de acesso associados, para além da lista com os dados específicos a cada um dos utilizadores é também possível notar na interface a presença de uma caixa de pesquisa que permite a procura de um utilizador através do seu nome – ou apenas parte do mesmo, como se vê na Imagem 49.

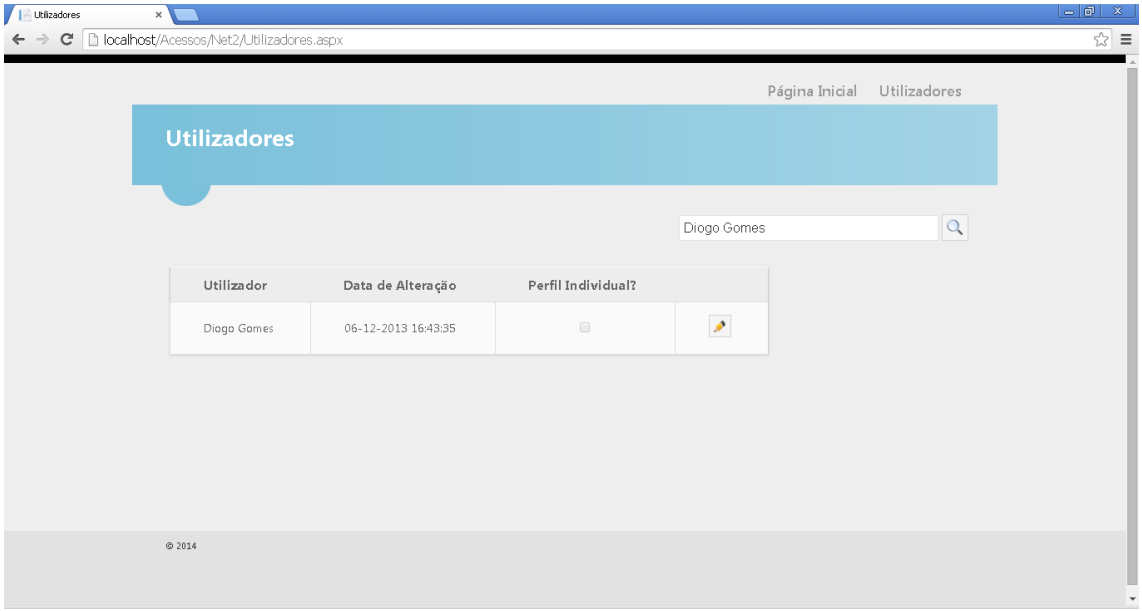

Imagem 49 – Interface web: pesquisa de utilizadores na plataforma

Relativamente a este objetivo a complexidade não se encontra ligada ao desenvolvimento da interface gráfica em si mas sim a todo o processamento interno realizado e ao estudo que envolveu a fase inicial desta implementação com a análise exaustiva dos dados que eram transmitidos entre o *software* atual de controlo de acessos e o servidor de base de dados configurado para o efeito.

Apesar de a estrutura da base de dados já alterada permitir o suporte da criação e gestão de perfis individuais, para que esta interface web funcionasse de acordo com o pretendido relativamente ao processo de atribuição de perfis individuais por utilizador, tiveram de ser feitas mais algumas modificações na base de dados, na sua maioria associadas à criação de um conjunto de procedimentos – *stored procedures*. O processo de criação destes procedimentos pode ser consultado no Anexo L – Stored procedures criados para a BD do Net2.

Nos pontos abaixo vão ser descritos de uma forma breve cada um dos procedimentos criados para que seja possível entender a necessidade a que os mesmos pretende dar resposta.

new_seleccionaUtilizadores: serve para processar a lista de utilizadores existentes na base de dados editada retornando-os com apenas alguns dos seus campos.

new_seleccionaUtilizador: procedimento semelhante ao anterior mas que retorna os dados de apenas um utilizador específico. Antes do retorno dos dados é verificado se esse utilizador tem perfil de acesso individual e ao mesmo tempo não tem permissões descritas. Em caso positivo associa esse utilizador ao perfil padrão que indica que esse utilizador não tem permissões associadas.

new_seleccionaAcessos: retorna uma lista com todos os acessos que estão associados a um determinado perfil.

new_seleccionaTimezones: retorna todas as possibilidades de horários existentes na base de dados para atribuição a um determinado perfil.

new_seleccionaAreas: retorna todas as áreas existentes na base de dados que podem ter um horário associado para cada perfil de controlo de acessos.

new_seleccionaAreasActivas: difere do procedimento anterior porque retorna apenas as áreas que se encontram ativas no sistema.

new_seleccionaDepartamentoPorUtilizador: retorna o nome do departamento a que um determinado utilizador está associado.

new_procuraUtilizador: retorna todos os dados de utilizadores que contêm no seu nome um valor especificado.

new_actualizaUtilizador: atualiza o registo de um utilizador de forma a conter a informação da data da última atualização.

new_criarPerfilIndividual: manipula os dados de acesso de um utilizador particular inserindo-o na tabela modificada anteriormente.

new_inserirAcessoIndividual: introduz uma nova permissão associada a um perfil já existente.

new_alterarAcessoIndividual: altera uma permissão já existente associada a um utilizador, enviando-lhe o horário e a área a atualizar.

new_removerAcessoIndividual: remove uma permissão existente associada a um utilizador, enviando-lhe a área para a qual deve ser retirado o acesso referente a esse utilizador específico.

Para além da implementação dos procedimentos acima descritos foi feita outra alteração à base de dados: a adição de um novo utilizador SQL que seria utilizado na comunicação entre a interface desenvolvida e o servidor SQL Net2 – ver na Imagem 50.

```
USE Net2;

CREATE LOGIN net2_usr WITH PASSWORD = 'net2_psw';
GO

CREATE USER net2_usr FOR LOGIN net2_usr;
GO

EXEC sp_addrolemember 'db_owner', 'net2_usr';
GO
```

Imagem 50 – BD Net2: criação de um utilizador para login na BD pela interface web

Terminadas as alterações necessárias à base de dados do servidor SQL do Net2 para permitir o correto funcionamento da interface web no âmbito do controlo e gestão dos perfis de acesso, configurou-se a solução web para que ficasse disponível na máquina virtual acedendo ao seu endereço IP de forma remota. Esse processo de configuração encontra-se no Anexo M – Configuração de "Internet Information Services".

6. Conclusões

6.1. Considerações Finais

A plataforma API – *Academic Playground & Innovation* – disponibiliza à comunidade académica um conjunto de webservices que podem ser utilizados nos mais diversos contextos a nível de projetos de unidades curriculares. Esta plataforma potencia a formação prática dos alunos uma vez que lhes permite aplicar os conceitos teóricos aprendidos em várias unidades curriculares relacionadas com este tema. É neste sentido que a divulgação e dinamização do portal é importante, de forma a tentar dar resposta a necessidades de alunos que frequentam cursos de outros departamentos para além do DETI, já que neste momento os conteúdos do portal estão mais direcionados para os cursos deste departamento. Assim sendo, a organização de um conjunto de concursos poderia ser um bom ponto de partida para colocar a comunidade académica num contacto mais direto com o portal API, ficando a conhecer os seus principais objetivos e conteúdos.

Relativamente ao processo de desenvolvimento de *webservices* um ponto a ter em atenção é a confirmação se os mesmos ficam devidamente documentados. Este procedimento faz com que a utilização dos *webservices* seja um processo mais simples, por exemplo, através da apresentação de casos de utilização ao longo da descrição das diferentes funcionalidades.

Durante o processo de desenvolvimento de *software* é importante garantir que a alteração de parte de um *webservice* não interfere com o bom funcionamento das restantes funcionalidades, principalmente quando existem dependências entre si. Assim sendo, é necessário implementar medidas para detetar inconsistências ou incompatibilidades resultantes de alterações, podendo para isso ser construídos testes de diferentes tipos, de entre eles os testes unitários. O grande objetivo deste tipo de testes é validar se as funções e métodos implementados estão sempre a funcionar da forma esperada.

É ainda importante mencionar a relevância do conceito de controlo de acessos que pode estar associado aos mais diversos tipos de objetivos, sendo os mais comuns o controlo de acessos a nível de elementos físicos ou a nível de segurança de informação. Para isso é necessária a existência de permissões de acesso que identificam as autorizações para os recursos devidos, podendo ser utilizados meios tecnológicos, que podem estar associados a elementos físicos – como cartões ou outro tipo de dispositivo móvel – que identificam a pessoa que está a aceder ao local e em que momento o está a fazer.

6.2. Trabalho Futuro

Durante o desenvolvimento de qualquer tipo de projeto é sempre possível fazer melhorias no trabalho existente e adicionar funcionalidades para acrescentar valor ao sistema. No contexto desta dissertação, o principal foco a nível de continuidade de trabalho existente seria, sem dúvida, a dinamização da plataforma.

Para dar maior visibilidade ao projeto e tentar promover a contribuição com algumas ideias por parte dos alunos poderiam ser afixados cartazes e distribuídos *flyers* em papel pela UA. Poderiam depois ser organizadas sessões conjuntas e presenciais para estabelecer contacto com os interessados em saber mais informações sobre a plataforma API e o projeto pensado para a sua dinamização.

Relativamente aos testes unitários, um ponto que poderia levar ao desenvolvimento de mais informação centra-se no estudo e comparação de ferramentas para integração contínua e implementação de testes unitários. Para além das ferramentas estudadas poderiam ser introduzidas mais algumas nesse estudo de forma a ser possível escolher um pequeno conjunto de entre as apresentadas para realizar os testes unitários. Desta forma, poderiam ser comparadas ferramentas a outros níveis, como por exemplo o seu desempenho.

Por último, deveriam ser acrescentadas melhorias na interface web relativa à componente de gestão de controlo de acessos. Estas incluem a adição de novas funcionalidades que permitissem a gestão de utilizadores, começando pela possibilidade de introduzir e manipular dados de novos utilizadores diretamente pela plataforma web. Ainda neste ponto pode referir-se a necessidade de realizar testes num ambiente real antes do seu enquadramento e integração no sistema de controlo de acessos já existente.

7. Bibliografia

[1] Wikipédia, “Sistema de informação,” [Online].

Available: http://pt.wikipedia.org/wiki/Sistema_de_informação.

[2] Wikipédia, “Database,” [Online].

Available: http://en.wikipedia.org/wiki/Database.

[3] Wikipédia, “Web service,” [Online].

Available: http://en.wikipedia.org/wiki/Web_service.

[4] DETI, IEETA e IT, “Academic Playground & Innovation,” [Online].

Available: http://api.web.ua.pt/.

[5] DETI, IEETA e IT, “Serviços da Universidade de Aveiro,” [Online].

Available: http://services.web.ua.pt/.

[6] D. F. CESAM, “Clim@UA,” [Online].

Available: http://climetua.fis.ua.pt/.

[7] Wikipédia, “Lightweight Directory Access Protocol,” [Online].

Available: http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol.

[8] Kioskea, “O protocolo LDAP,” [Online].

Available: http://pt.kioskea.net/contents/271-o-protocolo-ldap.

[9] Kioskea, “LDAP - O modelo de informação,” [Online].

Available: http://pt.kioskea.net/contents/877-ldap-o-modelo-de-informacao.

[10] ApacheFriends, “XAMPP,” [Online].

Available: https://www.apachefriends.org/index.html.

[11] Wikipédia, “XAMPP,” [Online].

Available: http://pt.wikipedia.org/wiki/XAMPP.

[12] Agil Open Source, “DokuWiki,” [Online].

Available: https://www.dokuwiki.org/dokuwiki.

[13] Wikipédia, “Unit testing,” [Online].

Available: http://en.wikipedia.org/wiki/Unit_testing.

[14] AnselmeIT, “O que é Teste Unitário?,” [Online].

Available: http://www.anselmeit.com/2011/05/o-que-e-teste-unitario.html.

[15] Wikipédia, “Continuous Integration,” [Online].

Available: http://en.wikipedia.org/wiki/Continuous_integration.

[16] M. Fowler, “Continuous Integration,” [Online].

Available: http://www.martinfowler.com/articles/continuousIntegration.html.

[17] Caelum, “Integração contínua e o processo Agile,” [Online].

Available: http://blog.caelum.com.br/integracao-continua/.

[18] [Online]. Available: https://mickaelistria.files.wordpress.com/2011/12/butler1000.jpg.

[19] Eclipse, “Meet Hudson,” [Online].

Available: http://wiki.eclipse.org/Hudson-ci/Meet_Hudson.

[20] W. S. E. Simon West, “Hudson - Your escape from "Integration Hell",” [Online].

Available: http://www.methodsandtools.com/tools/tools.php?hudson.

[21] T. O. Manfred Moser, “The Hudson Book,” [Online].

Available: http://www.eclipse.org/hudson/the-hudson-book/book-hudson.pdf.

[22] [Online]. Available: https://wiki.jenkins-ci.org/download/attachments/2916393/logo-title.png.

[23] C. Orr, “Meet Jenkins,” [Online].

Available: https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins.

[24] CloudBees, “About Jenkins CI,” [Online].

Available: https://www.cloudbees.com/jenkins/about.

[25] P. K. Kohsuke Kawaquchi, “Why choose Jenkins?,” [Online].

Available: https://wiki.jenkins-ci.org/pages/viewpage.action?pageId=53608972.

[26] CloudBees, “Five reasons why developers choose Jenkins over Hudson for continuous integration,” [Online].

Available: http://blog.cloudbees.com/2011/07/five-reasons-why-developers-choose.html.

[27] B. Bickel, “Jenkins vs Hudson - Time to upgrade!,” [Online].

Available: http://bobbickel.blogspot.pt/2011/03/jenkins-vs-hudson-time-to-upgrade.html.

[28] [Online]. Available: http://cruisecontrol.sourceforge.net/banner.png.

[29] CruiseControl, “CruiseControl,” [Online].

Available: http://cruisecontrol.sourceforge.net/.

[30] Wikipédia, “CruiseControl,” [Online].

Available: http://en.wikipedia.org/wiki/CruiseControl.

[31] CruiseControl, “Configuration Reference,” [Online].

Available: http://cruisecontrol.sourceforge.net/main/configxml.html.

[32] [Online]. Available: http://continuum.apache.org/images/continuum_logo_75.gif.

[33] Apache, “Apache Continuum,” [Online].

Available: http://continuum.apache.org/.

[34] B. Porter, “What is Continuum?,” [Online].

Available: https://cwiki.apache.org/confluence/display/CONTINUUM/Index.

[35] Wikipedia, “Apache Continuum,” [Online].

Available: http://en.wikipedia.org/wiki/Apache_Continuum.

[36] D. Eriksson, “What is Continuum and how do I install it?,” [Online].

Available: http://www.avajava.com/tutorials/lessons/what-is-continuum-and-how-do-i-install-it.html.

[37] D. Eriksoon, “How do I add a maven project to Continuum?,” [Online].

Available: http://www.avajava.com/tutorials/lessons/how-do-i-add-a-maven-project-to-continuum.html.

[38] JavaThoughts, “Hudson vs Continuum,” [Online].

Available: http://javathought.wordpress.com/2009/04/23/hudson-vs-continuum/.

[39] J. Ferguson, “Which open source CI tool is best suited for your application's environment?,” [Online].

Available: http://www.javaworld.com/article/2076188/build-ci-sdlc/which-open-source-ci-tool-is-best-suited-for-your-application-s-environment-.html.

[40] M. Pichler, “phpUnderControl - Continuous integration for PHP,” [Online].

Available: http://phpundercontrol.org/.

[41] D. Perrett, “Continuous Integration for PHP with phpUnderControl,” [Online].

Available: http://www.daveperrett.com/articles/2011/05/13/continuous-integration-for-php-with-php-under-control/.

[42] Jpablobr, “PHP staging environment for continuous integration,” [Online].

Available: http://jpablobr.com/past/php-staging-environment-for-continuous-integration-part-2.

[43] IamTope, “How to install CruiseControl,” [Online].

Available: http://www.iamtope.com/2010/04/how-to-install-cruisecontrol.html.

[44] Wikipédia, “Access Control,” [Online].

Available: http://en.wikipedia.org/wiki/Access_control.

[45] AskUbuntu, “What does "sudo apt-get update" do?,” [Online].

Available: http://askubuntu.com/questions/222348/what-does-sudo-apt-get-update-do.

[46] HelpUbuntu, “To install the default LAMP stack in Ubuntu 10.04 and above,” [Online].

Available: https://help.ubuntu.com/community/ApacheMySQLPHP.

[47] Microsoft, “Start SQL Server in Single-User Mode,” [Online].

Available: http://msdn.microsoft.com/en-us/library/ms188236.aspx.

8. Anexos

Anexo A – Conteúdos do portal Academic Playground & Innovation

Imagem 51 – API: página principal da plataforma

Imagem 52 – API: estrutura da plataforma

Imagem 53 – API: detalhes associados a um webservice

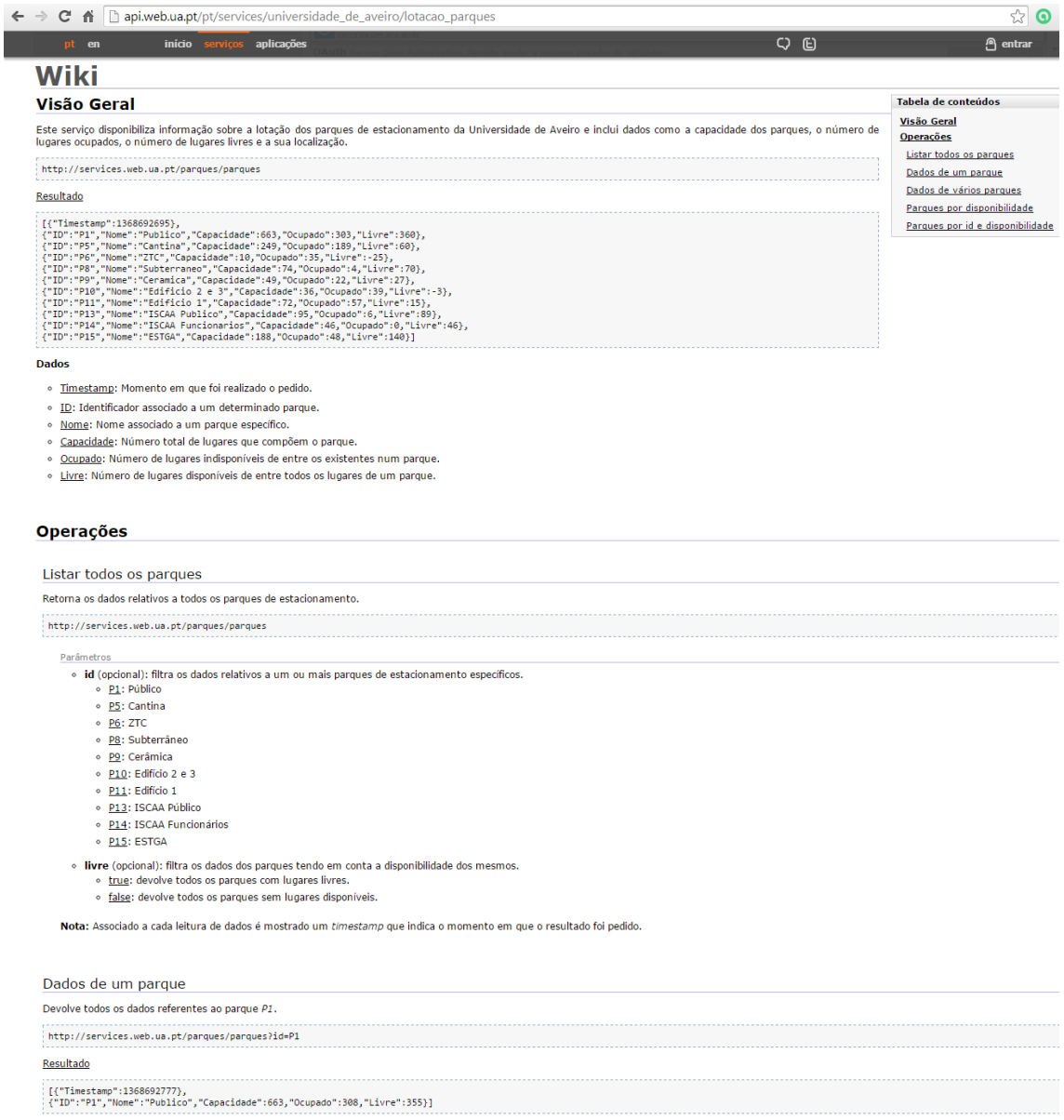

Imagem 54 – API: documentação de um webservice

Anexo B – Conteúdos do portal Serviços da Universidade de Aveiro

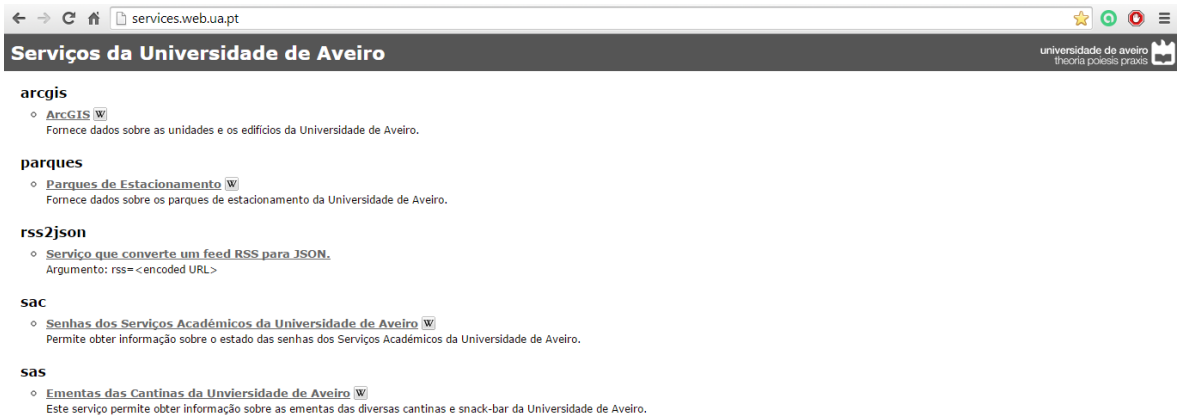

Imagem 55 – Serviços da UA: página principal do portal

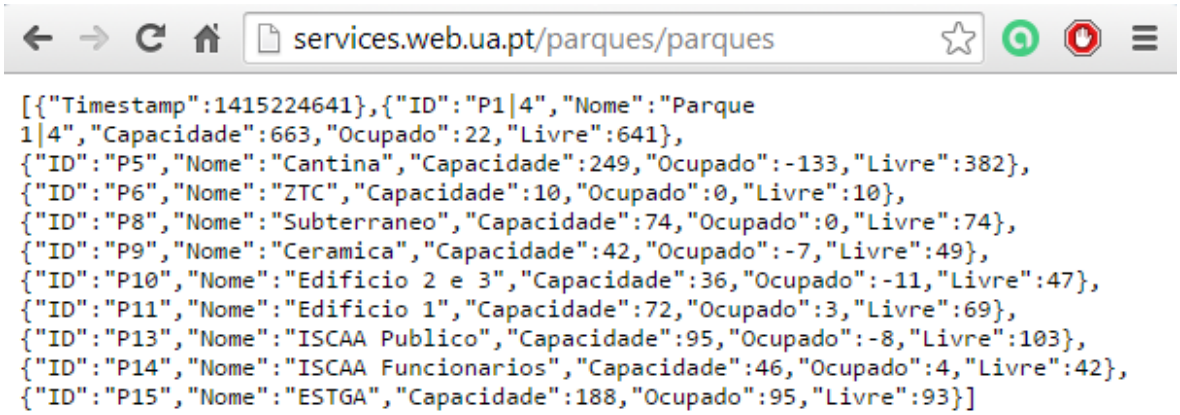

Imagem 56 – Serviços da UA: exemplo de dados de um webservice

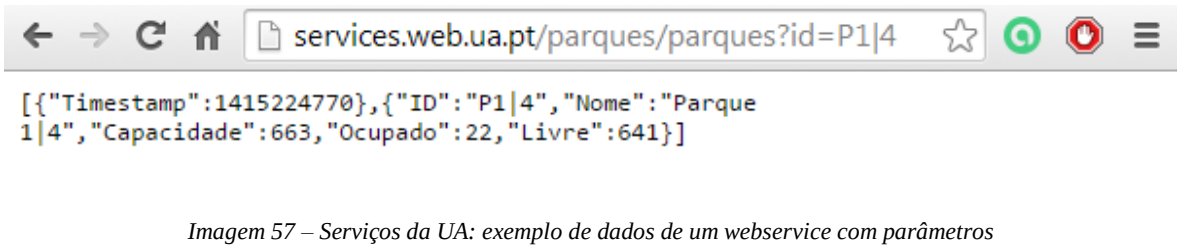

Imagem 57 – Serviços da UA: exemplo de dados de um webservice com parâmetros

Anexo C – Flyer enviado para divulgação da plataforma API

Imagem 58 – API: flyer enviado para divulgação da plataforma API

Anexo D – Script de criação da BD do webservice de assinaturas

```sql

-- Base de Dados: `db_perfis`

-- Estrutura da tabela `token`
CREATE TABLE IF NOT EXISTS `token` (
  `id_token` int(11) NOT NULL AUTO_INCREMENT,
  `fk_utilizador` int(11) NOT NULL,
  `token` varchar(20) NOT NULL,
  `api` int(5) NOT NULL,
  `activo` tinyint(1) NOT NULL,
  `visualizacoes` int(11) NOT NULL,
  `descricao` longtext NOT NULL,
  PRIMARY KEY (`id_token`),
  UNIQUE KEY `token` (`token`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=6 ;

-- Estrutura da tabela `utilizador`
CREATE TABLE IF NOT EXISTS `utilizador` (
  `id_utilizador` int(11) NOT NULL AUTO_INCREMENT,
  `email` varchar(50) NOT NULL,
  `acesso` tinyint(1) NOT NULL,
  PRIMARY KEY (`id_utilizador`),
  UNIQUE KEY `email` (`email`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=4 ;

ALTER TABLE `token` ADD CONSTRAINT `fk_utilizador` FOREIGN KEY token(fk_utilizador) REFERENCES utilizador(id_utilizador)

```

Imagem 59 – db_perfis: script de criação da BD do webservice de assinaturas

	#	Nome	Tipo
☐	1	id_utilizador	int(11)
☐	2	email	varchar(50)
☐	3	acesso	tinyint(1)

Imagem 60 – db_perfis: estrutura gráfica da tabela "utilizador"

	#	Nome	Tipo
☐	1	id_token	int(11)
☐	2	fk_utilizador	int(11)
☐	3	token	varchar(20)
☐	4	api	int(5)
☐	5	activo	tinyint(1)
☐	6	visualizacoes	int(11)
☐	7	descricao	longtext

Imagem 61 – db_perfis: estrutura gráfica da tabela "token"

Anexo E – Documentação do webservice de assinaturas

Após o desenvolvimento do *webservice* de assinaturas descrito ao longo do capítulo 3 – `Desenvolvimento de um webservice` – foi criada uma página de documentação para o mesmo na plataforma *Academic Playground & Innovation*.

Esta plataforma tem integrada a *DokuWiki* que é uma ferramenta que permite criar e editar páginas de documentação de forma rápida. A utilização da *DokuWiki* facilita a gestão da documentação devido principalmente à sua sintaxe simples e à existência de controlo de versões das páginas criadas. **[44]** É importante mencionar que a configuração do ambiente da *wiki* na plataforma API é anterior à minha interação com o portal.

Mesmo sem efetuar login, ao abrir a página principal da plataforma API, é possível ver uma parte das suas funcionalidades – como a listagem de todos os *webservices* públicos e algumas aplicações desenvolvidas no âmbito do portal. A partir da página principal é ainda possível seguir a ligação para o *Blog* ou para o *Twitter* da plataforma API.

Após inserir as credenciais de acesso ao portal são apresentadas mais funcionalidades e, no caso de o utilizador ter permissões para documentação de *webservices*, é apresentado no menu inicial o ponto de acesso à wiki inerente à plataforma. As `Imagens 62 e 63` ilustram a diferença entre os dois casos referidos.

Imagem 62 – API: menu de navegação sem login efetuado

Imagem 63 – API: menu de navegação com login efetuado e permissão de edição da wiki

Antes de iniciar a construção da *wiki* em si é necessário fazer a indicação na plataforma de que existe um novo serviço web, sendo indicado o nome do serviço a adicionar e uma pequena descrição do mesmo. Além disso, é ainda permitido selecionar a opção de tornar o serviço privado ou não visível.

Um *webservice* privado é aquele que necessita que um utilizador faça *login* na plataforma API com o seu utilizador universal antes de ter acesso à documentação; um serviço não visível é aquele que, para além de ser privado, ainda não é apresentado aos utilizadores. Esta situação é comum durante a fase de desenvolvimento da documentação, para que esta não seja acedida numa altura em que ainda se encontra incompleta ou com necessidade de verificações. É permitido ver a possibilidade de escolha do tipo de serviço na Imagem 64.

Imagem 64 – API: descrição de um novo webservice

Após a adição da secção de documentação à plataforma API para um novo *webservice* é mostrada a mensagem "Esta wiki ainda não existe", que será permanente até a página de documentação ser criada, como se pode ver na Imagem 65.

Imagem 65 – Wiki: novo webservice sem documentação

Desta forma, com a finalidade de introduzir conteúdos a mostrar na página da *wiki* de um *webservice* acabado de criar, é necessário aceder ao separador `Wiki` que nos apresenta a página ilustrada na `Imagem 66`.

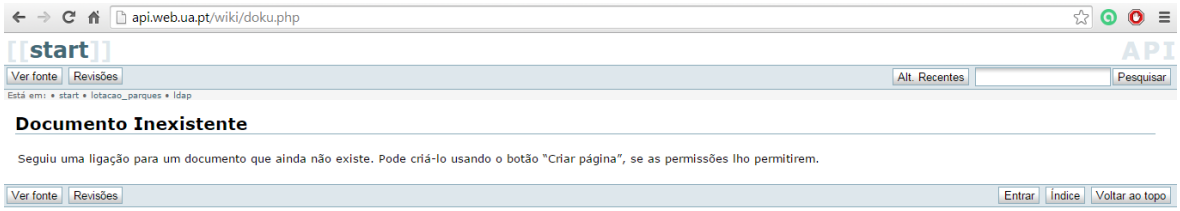

Imagem 66 – Wiki: página inicial da DokuWiki

A ferramenta de *wiki* integrada na plataforma API – designada por *DokuWiki* – tem uma componente de autenticação separada da restante plataforma, sendo para isso necessária a autenticação prévia na mesma com credenciais que tenham associadas permissões de construção de páginas. As credenciais não precisam de estar necessariamente associadas a um utilizador universal da Universidade de Aveiro.

O processo de documentação não é complexo, o único problema verificado – ocorrido na altura da bolsa de integração na investigação – foi precisamente fazer as configurações da conta a ter permissões de edição de ficheiros e para que este conjunto de funcionalidades da *wiki* estivessem disponíveis para essa conta sem restrições.

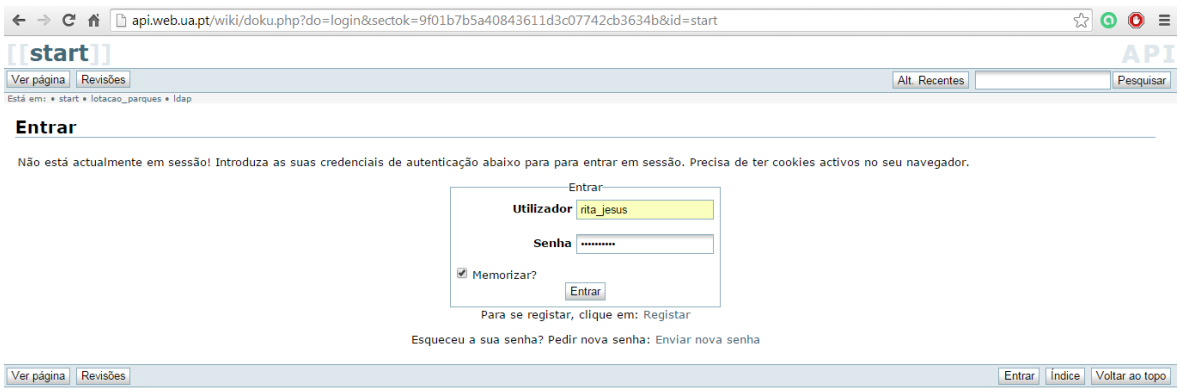

Imagem 67 – Wiki: autenticação para edição da wiki

Após a autenticação na ferramenta da Wiki – como se vê na `Imagem 67` – é possível consultar um índice que nos mostra as páginas já existentes relativas a documentação de *webservices* da plataforma API – ver `Imagem 68`.

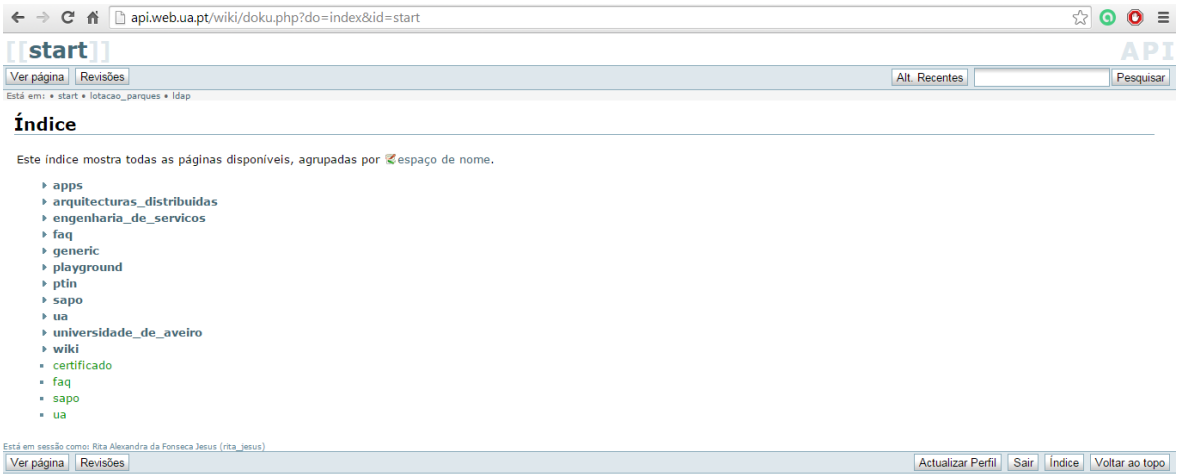

Imagem 68 – Wiki: lista de páginas de documentação de webservices

O *webservice* de assinaturas fica por pré-definição na secção da Universidade de Aveiro com o nome Assinaturas. Posto isto, para criar a página de documentação bastou aceder a `http://api.web.ua.pt/wiki/doku.php?id=universidade_de_aveiro:assinaturas`. No início, a página tem o aspeto da `Imagem 69`.

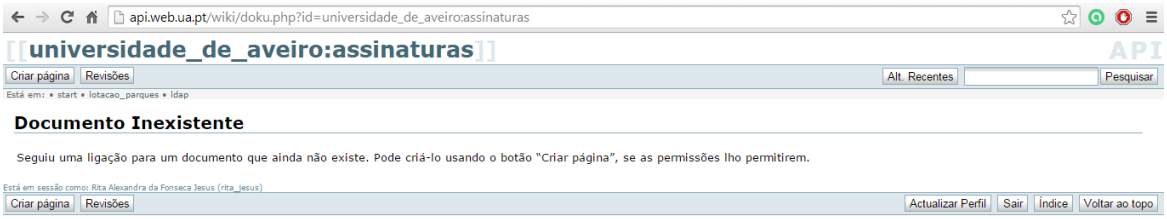

Imagem 69 – Wiki: página da wiki para o novo webservice

Neste momento foi apresentado um editor de texto onde foi possível introduzir o código de produção da documentação do *webservice* com algumas formatações específicas da *wiki*. Esta documentação ficava automaticamente visível na plataforma API, contendo detalhes de cada funcionalidade particular do serviço web – como se vê na `Imagem 70`.

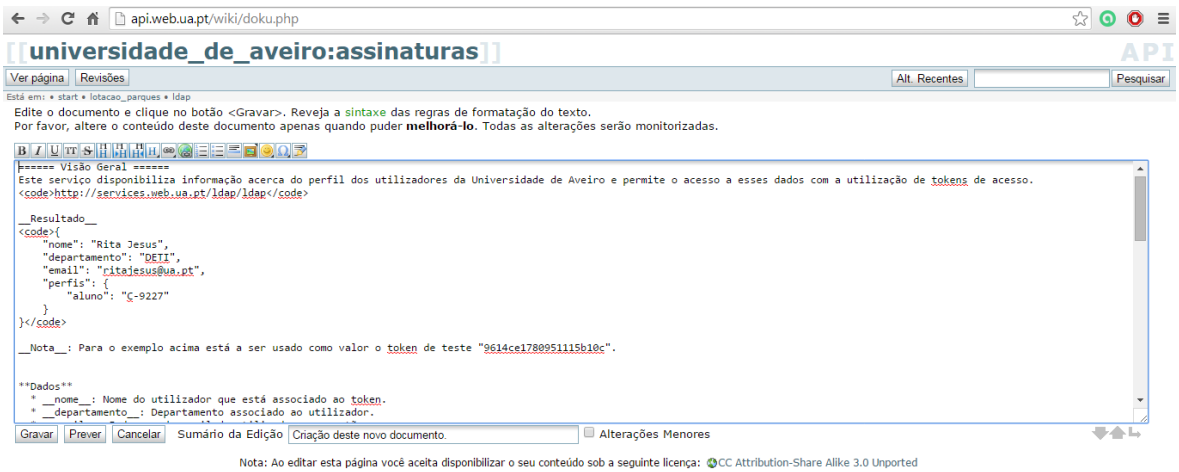

Imagem 70 – Wiki: página de documentação do novo serviço em construção

Após a introdução do código necessário para a documentação ficar completa foi possível guardar finalmente a página para que as alterações tivessem efeito, a nível de *DokuWiki* – como se vê na `Imagem 71` – e na plataforma API.

Imagem 71 – Wiki: página de documentação do novo serviço após adicionada

Após o processo de construção da página de documentação ter sido terminado, foi logo possível consultar, no portal API, a secção de documentação previamente desenvolvida, estando obviamente associada ao *webservice* implementado. A página de documentação do serviço web de assinaturas – ilustrada pela `Imagem 72` – está dividida em duas partes essenciais, sendo a primeira designada por Visão Geral. Nesta secção é feita uma pequena descrição do *webservice* e indicado o endereço onde o mesmo pode ser encontrado. Para além disso, é apresentado um exemplo de resultado a uma chamada do *webservice* para que seja possível mostrar quais os dados recebidos e a sua estrutura.

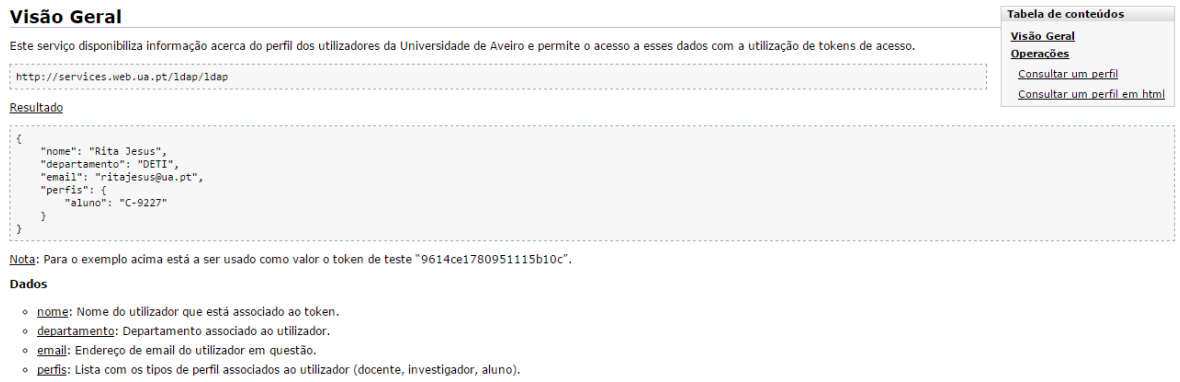

Imagem 72 – API: documentação do serviço de assinaturas – Visão geral

Após dada uma visão geral do sistema é apresentada a segunda secção da documentação, designada por Operações. Tal como o próprio nome indica, esta secção é constituída pela descrição do conjunto de funcionalidades disponibilizadas pelo webservice. Na Imagem 73 é possível ver a forma como a informação é apresentada para uma das suas operações – neste caso, a consulta de um perfil de utilizador.

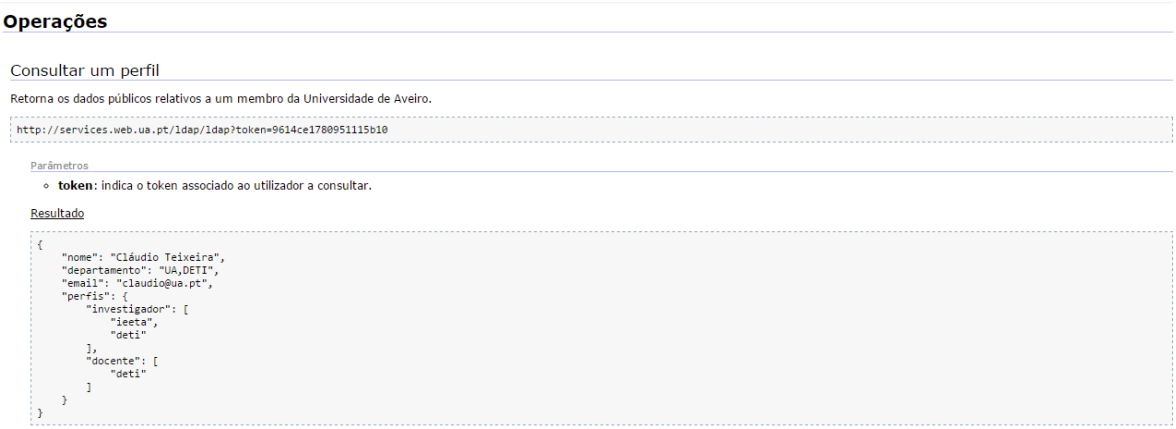

Imagem 73 – API: documentação do serviço de assinaturas – Operações: consulta em JSON

Apesar de a informação numa consulta de um perfil ser mostrada em formato JSON é também possível consultar essa informação em HTML. À semelhança do que acontece para a função anterior, é apresentada uma breve explicação do seu objetivo e mostrado um exemplo de utilização – como se pode ver na Imagem 74. Aqui são indicados os parâmetros obrigatórios e opcionais a usar na chamada da função do *webservice*, diferenciando de forma evidente os dois tipos de parâmetros.

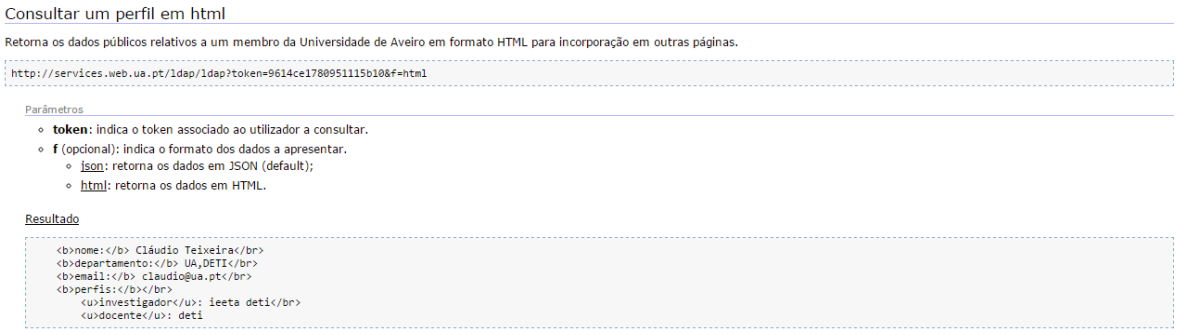

Imagem 74 – API: documentação do serviço de assinaturas – Operações: consulta em HTML

Anexo F – Instalação do ambiente de testes CruiseControl

Antes de dar início ao processo de instalação foi executado o comando `apt-get update` para que fossem transferidas as listas com as versões de pacotes mais recentes existentes nos repositórios. **[45]**

Seguidamente procedeu-se à instalação do Java com o comando `apt-get install default-jre` e verificou-se o sucesso da sua instalação através do comando `java -version` que mostra qual a versão mais recente do Java instalada no sistema. A instalação do Java é necessária para o funcionamento do *CruiseControl*.

```
java version "1.7.0_65"
OpenJDK Runtime Environment (IcedTea 2.5.2) (7u65-2.5.2-3~14.04)
OpenJDK 64-Bit Server VM (build 24.65-b04, mixed mode)
```

Imagem 75 – CruiseControl: versão de instalação do Java

Após isso foram instalados alguns pacotes de PHP essenciais para o funcionamento do phpUnderControl, executando o comando `apt-get install php5-dev php-pear php5-xdebug`. Para confirmar a correta instalação do Xdebug foi executado o comando `vim /etc/php5/mods-available/xdebug.ini` para confirmar as configurações.

```
zend_extension=/usr/lib/php5/20121212/xdebug.so
xdebug.profiler_enable_trigger = onc
```

Imagem 76 – CruiseControl: instalação do Xdebug

Para utilizar o *CruiseControl* e *phpUnderControl* precisamos de um servidor local, para isso executamos o comando `apt-get install lamp-server^` que inclui Apache2, PHP5, e MySQL5.5. **[46]** Durante a instalação foram necessárias algumas configurações relativas ao mysql-server-5.5, mais especificamente a introdução e confirmação de uma palavra-passe para o root (utilizador de administração). Após a conclusão da instalação do lamp-server o servidor apache2 já se encontrava ativo no localhost.

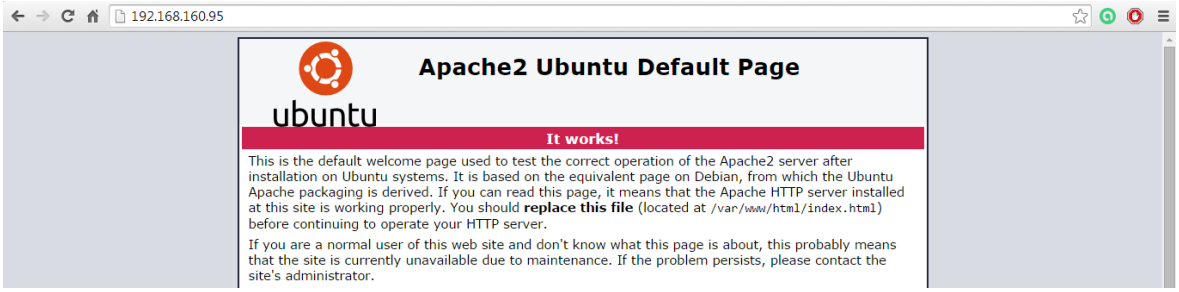

Imagem 77 – CruiseControl: servidor de apache no localhost

Em seguida foi criado um ficheiro na pasta do projeto a correr no servidor local através do comando `touch /var/www/html/phpinfo.php` e editado com o comando `vim /var/www/html/phpinfo.php` para introduzir o conteúdo da imagem abaixo.

```
<?php phpinfo(); ?>
```

Imagem 78 – CruiseControl: conteúdo phpinfo()

Ao abrir no servidor local a página criada anteriormente são apresentadas todas as informações associadas à instalação e respetivas configurações do PHP. Esta ação foi realizada com o objetivo de verificar a instalação do Xdebug associada ao PHP.

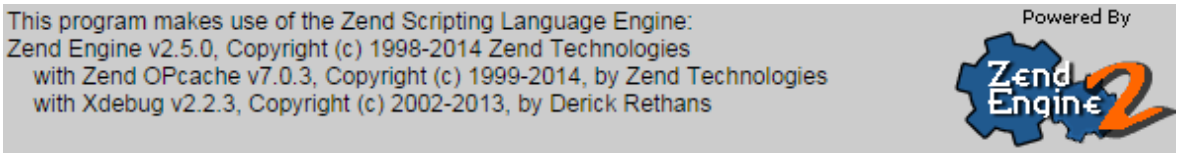

Imagem 79 – CruiseControl: versão de instalação do Xdebug

A preparação do ambiente de testes propriamente dito começou com a instalação do *CruiseControl*. O primeiro comando executado foi `cd /opt` para que a sua instalação fosse realizada nesse diretório, seguindo-se da transferência de uma pasta zipada com todos os ficheiros referentes aos seus conteúdos através da execução do comando `wget http://heanet.dl.sourceforge.net/sourceforge/cruisecontrol/cruisecontrol-bin-2.8.4.zip`.

Depois disso foi instalado o *unzip* com o comando `apt-get install unzip` para ser feita a extração dos conteúdos da pasta transferida executando `unzip cruisecontrol-bin-2.8.4.zip`. Para ser possível um acesso mais fácil à pasta do *CruiseControl* foi criado um atalho para a mesma através do comando `ln -s cruisecontrol-bin-2.8.4 cruisecontrol`. Após isso foi removida a pasta zipada executando `rm cruisecontrol-bin2.8.4.zip`.

Neste momento ainda não era possível executar o *CruiseControl* uma vez que não existia um script de inicialização do mesmo. Para combater esta falha foi criado um ficheiro com o comando `touch /etc/init.d/cruisecontrol`, editado em seguida executando `vim /etc/init.d/cruisecontrol` **[41]** e tornado executável através do comando `chmod a+x /etc/init.d/cruisecontrol`.

```
#!/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
. /lib/lsb/init-functions
JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64/
NAME=cruisecontrol
DAEMON=/opt/cruisecontrol/cruisecontrol.sh
PIDFILE=/opt/cruisecontrol/cc.pid

test -x $DAEMON || exit 5

RUNASUSER=www-data
UGID=$(getent passwd $RUNASUSER | cut -f 3,4 -d:) || true

case $1 in
start)
log_daemon_msg "Starting Cruisecontrol server" "cc"
if [ -z "$UGID" ]; then
log_failure_msg "user \"$RUNASUSER\" does not exist"
exit 1
fi
cd /opt/cruisecontrol/
./cruisecontrol.sh > /dev/null 2>&1
log_end_msg $?
;;
stop)
log_daemon_msg "Stopping Cruisecontrol server" "cc"
start-stop-daemon --stop --quiet --oknodo --pidfile $PIDFILE
log_end_msg $?
rm -f $PIDFILE
;;
restart|force-reload)
$0 stop && sleep 2 && $0 start
;;
status)
pidofproc -p $PIDFILE $DAEMON >/dev/null
status=$?
if [ $status -eq 0 ]; then
log_success_msg "Cruisecontrol server is running."
else
log_failure_msg "Cruisecontrol server is not running."
fi
exit $status
;;
*)
echo "Usage: $0 {start|stop|restart|force-reload|status}"
exit 2
;;
esac
```

Imagem 80 – CruiseControl: script de inicialização do CruiseControl

Para que o *CruiseControl* fosse iniciado automaticamente com a ligação do servidor foi usado o comando `update-rc.d cruisecontrol defaults`. Foi também necessário editar um dos ficheiros do CruiseControl para alterar o caminho do Java para uma versão mais recente usando `vim /opt/cruisecontrol/cruisecontrol.sh`.

```
#!/usr/bin/env bash
JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64/
```

Imagem 81 – CruiseControl: alteração da JAVA_HOME

Após todas as alterações mencionadas foi possível iniciar o *CruiseControl* de forma a ficar a correr no servidor, para isso foi usado o comando `/etc/init.d/cruisecontrol start`. Depois de iniciar o *CruiseControl* este passou a estar acessível através do endereço `192.168.160.95:8080/cruisecontrol`.

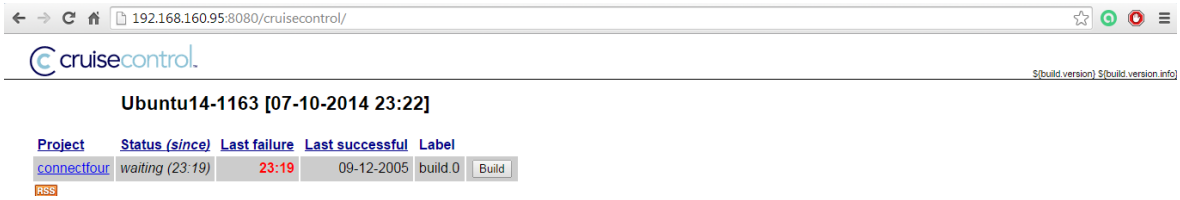

Imagem 82 – CruiseControl: página inicial do CruiseControl

Depois de completamente instalado e configurado o *CruiseControl* foi necessário proceder à instalação do *phpUnderControl*. Este requer algumas dependências que foram instaladas executando cada um dos comandos da lista que se segue.

```
pear upgrade pear
pear channel-discover components.ez.no
pear install -a ezc/Graph
pear config-set preferred_state beta
pear channel-discover pear.phpunit.de
pear channel-discover pear.symfony-project.com
pear channel-discover pear.phpundercontrol.org
pear install --alldeps --force phpuc/phpUnderControl-beta
pear channel-discover pear.pdepend.org
pear install channel://pear.pdepend.org/PHP_Depend-0.9.11
pear channel-discover pear.phpmd.org
pear install channel://pear.phpmd.org/PHP_PMD-0.2.4
pear install --alldeps phpunit/phpcpd
pear channel-discover pear.phing.info
pear config-set preferred_state alpha
pear install --alldeps --force phing/phing
```

De entre as dependências acima deve ser reforçada a importância do *PHPUnit* usada pelo *phpUnderControl* ao longo do desenvolvimento dos testes.

Após concluída a instalação do *phpUnderControl* foi possível fazer a instalação de um conjunto de testes exemplo no projeto *CruiseControl* previamente configurado. Para isso bastou executar o comando `phpuc install /opt/cruisecontrol` e reiniciar o *CruiseControl* da seguinte forma `/etc/init.d/cruisecontrol restart`.

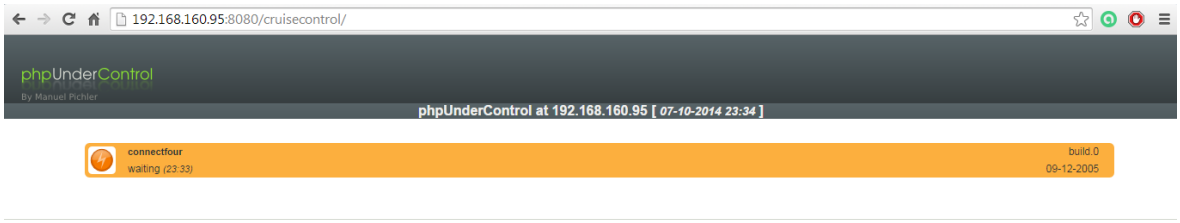

Imagem 83 – CruiseControl: página inicial do phpUnderControl

O exemplo criado tinha o nome `connectfour` e pretendia retratar o tradicional jogo 4-em-linha. Neste momento o projeto exemplo aparecia com um fundo alaranjado, significando que algo não estava correto na configuração do mesmo. Isto acontecia porque ainda não tinha sido instalado o *Ant* nem o *JUnit*.

O *Ant* foi instalado através da execução do comando `apt-get install ant`, seguido pela instalação de uma dependência Java necessária ao seu funcionamento usando `apt-get install openjdk-7-jdk`. Para confirmar a correta instalação do *Ant* foi executado o comando `ant -version`.

```
Apache Ant(TM) version 1.9.3 compiled on April 8 2014
```

Imagem 84 – CruiseControl: versão de instalação do Ant

O *JUnit* foi instalado com o comando `apt-get install junit`, seguido do comando `junit -v` para confirmação da sua instalação.

```
JUnit 3.8.2 by Kent Beck and Erich Gamma
Usage: TestRunner [-wait] testCaseName, where name is the name of the TestCase cl
ass
```

Imagem 85 – CruiseControl: versão de instalação do JUnit

Após isto ainda foi necessária a edição das variáveis de ambiente para incluir o caminho do *Java*. Tal pôde ser feito através da execução do comando `vim /etc/environment` adicionando-lhe a segunda linha visível na imagem abaixo.

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us
r/local/games"
JAVA_HOME="/usr/lib/jvm/java-7-openjdk-amd64"
```

Imagem 86 – CruiseControl: adição da variável de ambiente JAVA_HOME

Para que todas as alterações mencionadas tivessem efeito foi necessário reiniciar o *CruiseControl* usando `/etc/init.d/cruisecontrol restart`. Neste momento já era possível ver o projeto connectfour sem o fundo alaranjado, significando que o mesmo já se encontrava completamente funcional.

Imagem 87 – CruiseControl: página inicial do CruiseControl + phpUnderControl

Clicando sobre o nome do projeto exemplo foi possível ver mais informações referentes à sua execução. De entre a informação apresentada podemos destacar o número da build atual e o número de testes unitários associados, todos eles executados com sucesso.

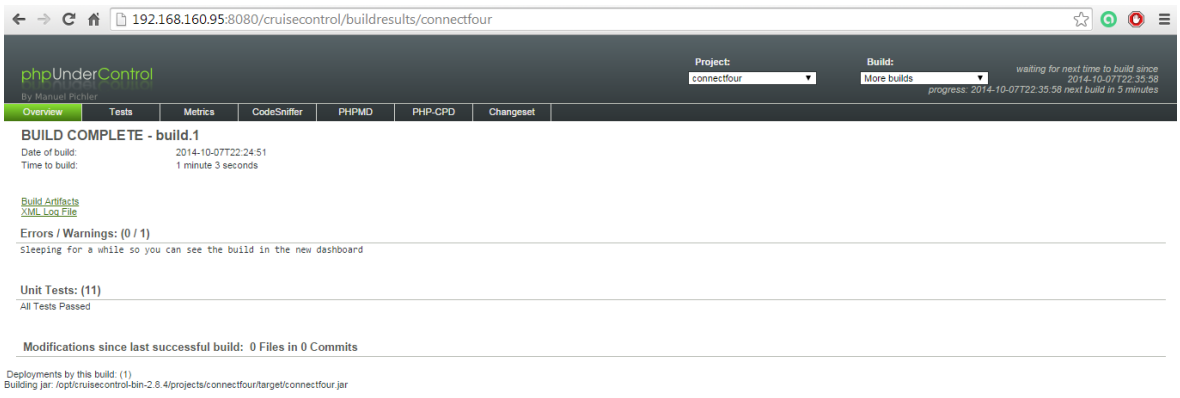

Imagem 88 – CruiseControl: detalhes de uma build de um projeto CruiseControl

Anexo G – Configuração do CruiseControl para o serviço de parques

Para que fosse possível a realização de testes unitários sobre o *webservice* de parques foi necessária a configuração de um projeto específico no ambiente de testes *CruiseControl* tendo por base a criação da seguinte estrutura de diretórios e ficheiros:

```
cd /opt/cruisecontrol/projects
mkdir ua_parques
touch ua_parques/build.xml
mkdir ua_parques/src
mkdir ua_parques/src/ua_parques
touch ua_parques/src/ua_parques/Parques.java
mkdir ua_parques/test
mkdir ua_parques/test/ua_parques
touch ua_parques/test/ua_parques/ParquesTest.java
```

Em seguida foi editado o ficheiro `Parques.java` cujo conteúdo pode ser visto na imagem abaixo. Este ficheiro era inicialmente constituído por uma única função `infoParques()` que retornava uma pequena informação acerca do *webservice*. É de notar que neste momento o projeto ainda não tinha qualquer associação com o *webservice* existente.

```
package ua_parques;

public class Parques {

        public static void main(String[] args) {
                Parques p = new Parques();
                System.out.println(p.infoParques());
        }

        public String infoParques() {
                return "Webservice -> Parques de Estacionamento";
        }
}
```

Imagem 89 – CruiseControl: ficheiro Parques.java - infoParques

Seguiu-se a edição do ficheiro `ParquesTest.java` cujo conteúdo é apresentado em seguida na imagem. Este ficheiro era também inicialmente constituído por uma única função `testInfoParques()` que pretendia verificar o retorno correto da mensagem enviada pela função `infoParques()` do ficheiro `Parques.java`. A esta validação está associada uma mensagem de erro a apresentar em caso de falha.

```java
package ua_parques;

import junit.framework.TestCase;

public class ParquesTest extends TestCase {
	private Parques p = new Parques();

	public void testInfoParques() {
		String msg_info = "ERROR[1]: infoParques()";
		String info = "Webservice -> Parques de Estacionamento";

		assertTrue(msg_info, p.infoParques().equals(info));
	}
}
```

Imagem 90 – CruiseControl: ficheiro ParquesTeste.java - testInfoParques

Para que fosse possível executar o projeto foi necessário editar o ficheiro de configuração `build.xml` cujo conteúdo é apresentado abaixo. Este ficheiro foi estruturado com base no ficheiro de configuração do projeto de exemplo `connectfour` do CruiseControl. O ficheiro `build.xml` serve para indicar ao *CruiseControl* de que forma deve ser feita a construção do projeto – também designada por *build* – sendo importante fazer uma pequena descrição de cada uma das configurações presentes:

- **all**: depende de todas as outras configurações e serve para iniciar uma nova *build*;
- **clean**: apaga o diretório `target` que contém os ficheiros gerados pela última *build*;
- **compile**: compila os ficheiros fonte Java para o diretório `target/classes`;
- **sleep**: mantém a *build* em espera 10 segundos para a sua construção ser notada;
- **test**: compila os ficheiros de teste Java para o diretório `target/test-classes` e configura a execução dos testes *JUnit* para o diretório `target/test-results`;
- **jar**: gera o ficheiro .jar referente ao projeto depois de uma nova *build* do mesmo.

```xml
<project name="ua_parques" default="all">
    <target name="all" depends="clean, compile, sleep, test, jar"/>

    <target name="clean">
        <delete dir="target" quiet="true" />
    </target>

    <target name="compile">
        <mkdir dir="target/classes"/>
        <javac srcdir="src" destdir="target/classes"/>
    </target>

    <target name="sleep">
        <sleep seconds="10" />
    </target>
```

Imagem 91 – CruiseControl: ficheiro build.xml do projeto ua_parques

```
    <target name="test" depends="compile">
        <mkdir dir="target/test-classes"/>
        <javac srcdir="test" destdir="target/test-classes">
            <classpath>
                <pathelement location="target/classes"/>
                <pathelement location="lib/junit.jar"/>
            </classpath>
        </javac>

        <mkdir dir="target/test-results"/>
        <junit haltonfailure="no" printsummary="on">
            <classpath >
                <pathelement location="target/classes"/>
                <pathelement location="lib/junit.jar"/>
                <pathelement location="target/test-classes"/>
            </classpath>
            <formatter type="brief" usefile="false"/>
            <formatter type="xml" />
            <batchtest todir="target/test-results" >
                <fileset dir="target/test-classes" includes="**/*Test.class"/>
            </batchtest>
        </junit>
    </target>

    <target name="jar" depends="compile">
        <jar jarfile="target/ua_parques.jar" basedir="target/classes"/>
    </target>
</project>
```

Imagem 92 – CruiseControl: ficheiro build.xml do projeto ua_parques (continuação)

Nesta fase o projeto estava devidamente configurado para integração no *CruiseControl*, estando apenas a faltar a edição do ficheiro de configuração `config.xml` presente no diretório `/opt/cruisecontrol/projects` que iria permitir saber ao *phpUnderControl* de que forma deveria ser realizada a sua *build*.

Este ficheiro foi editado com base nas configurações apresentadas para o projeto de exemplo `connectfour` do *CruiseControl*. Nessas configurações do projeto é indicado que as informações da *build* atual devem ser armazenadas num ficheiro `status.txt` e que antes de cada *build* devem ser eliminados os ficheiros de teste gerados para a *build* anterior.

É também indicado que são procurados apenas os ficheiros com data de edição inferior a 30 segundos para verificação de alterações e que estão agendadas novas *builds* para este projeto a cada 300 segundos – o equivalente a 5 minutos.

É ainda referido que se encontram *logs* das *builds* na pasta `target/test-results` e que o ficheiro .jar se encontra na pasta `target`.

```
<project name="ua_parques">
    <listeners>
      <currentbuildstatuslistener file="logs/${project.name}/status.txt"/>
    </listeners>
    <bootstrappers>
      <antbootstrapper anthome="apache-ant-1.7.0" buildfile="projects/${project.n
ame}/build.xml" target="clean"/>
    </bootstrappers>
    <modificationset quietperiod="30">
      <filesystem folder="projects/${project.name}"/>
    </modificationset>
    <schedule interval="300">
      <ant anthome="apache-ant-1.7.0" buildfile="projects/${project.name}/build.x
ml"/>
    </schedule>
    <log>
      <merge dir="projects/${project.name}/target/test-results"/>
    </log>
    <publishers>
      <onsuccess>
        <artifactspublisher dest="artifacts/${project.name}" file="projects/${pro
ject.name}/target/${project.name}.jar"/>
      </onsuccess>
    </publishers>
  </project>
```

Imagem 93 – CruiseControl: ficheiro config.xml do projeto ua_parques

Depois de guardadas as alterações aos ficheiros `Parques.java`, `ParquesTest.java`, `build.xml` e `config.xml` é possível aceder ao conjunto de projetos do *CruiseControl* no endereço `192.168.160.95:8080/cruisecontrol`.

Numa primeira fase é visto o projeto no estado `building`, seguindo-se o estado `waiting` que significa que a *build* foi criada e aguarda uma nova *build*. Tal pode acontecer porque passaram 5 minutos desde a última *build*, porque foram editados ficheiros do projeto ou porque foi forçada a criação de uma nova *build*.

Imagem 94 – CruiseControl: projeto em estado de construção

Imagem 95 – CruiseControl: projeto em estado de espera

Assim que a *build* é gerada é possível consultar os dados referentes à mesma. Neste caso dá para ver que os dados apresentados são referentes à primeira *build* do projeto e que existe apenas um teste unitário a ser executado – sendo executado com sucesso.

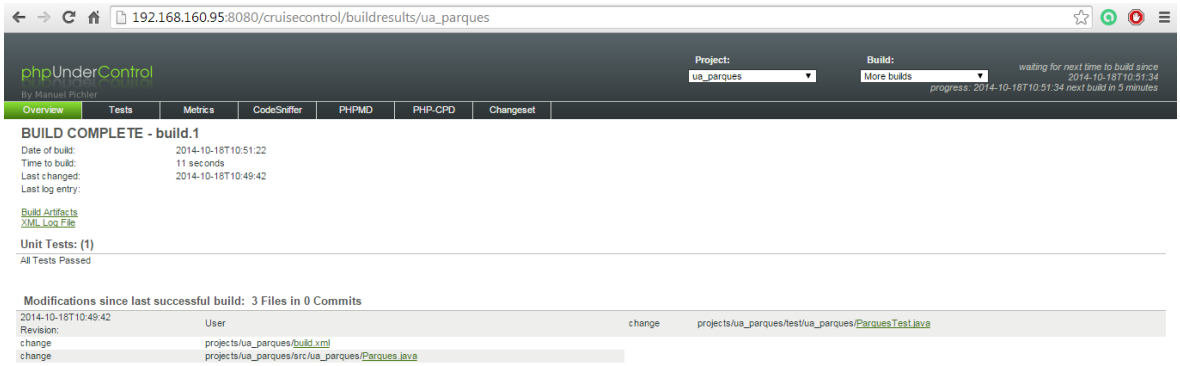

Imagem 96 – CruiseControl: projeto construído com sucesso

Apenas com o intuito de ver as diferenças apresentadas quando a construção de uma nova *build* do projeto falha foi editado o ficheiro `ParquesTest.java` para ficar com o aspeto apresentado abaixo. Foi alterada na função `testInfoParques()` a frase a comparar com a informação que é retornada pela função `infoParques()` do ficheiro `Parques.java`. Desta forma os dados comparados serão diferentes e a realização do teste unitário falhará.

```
package ua_parques;

import junit.framework.TestCase;

public class ParquesTest extends TestCase {
	private Parques p = new Parques();

	public void testInfoParques() {
		String msg_info = "ERROR[1]: infoParques()";
		String infoWRONG = "Parques de Estacionamento";

		assertTrue(msg_info, p.infoParques().equals(infoWRONG));
	}
}
```

Imagem 97 – CruiseControl: ficheiro ParquesTeste.java configurado com ero

É possível verificar pela imagem seguinte que neste momento a *build* do projeto mostra um caso de falha nos testes unitários existentes.

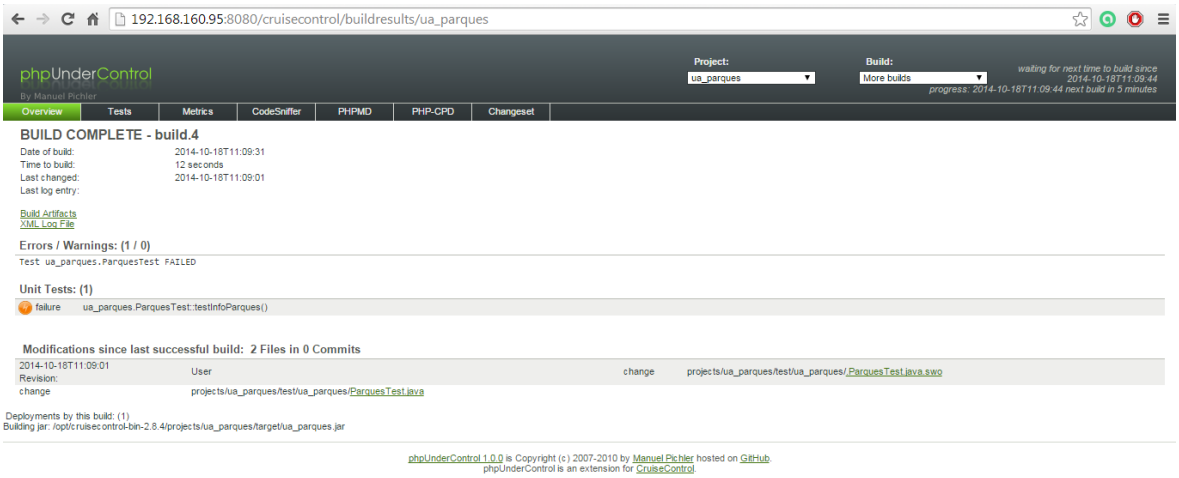

Imagem 98 – CruiseControl: projeto construído com erro

Se a compilação e validação dos testes unitários for executada através da linha de comandos usando `ant test` na pasta onde se encontra o ficheiro `build.xml` é possível ter a perceção das pastas que vão sendo criadas, das compilações de ficheiros que vão sendo feitas e ainda de dados associados ao *JUnit*. A maior diferença de apresentação de dados entre este método e o anterior é que neste caso é apresentada a mensagem de erro associada à falha do teste. Esta mensagem é útil para entender em que validação específica o teste falhou caso existam múltiplas validações dentro da mesma função.

```
Buildfile: /opt/cruisecontrol-bin-2.8.4/projects/ua_parques/build.xml

compile:
    [mkdir] Created dir: /opt/cruisecontrol-bin-2.8.4/projects/ua_parques/target/
classes
    [javac] /opt/cruisecontrol-bin-2.8.4/projects/ua_parques/build.xml:10: warnin
g: 'includeantruntime' was not set, defaulting to build.sysclasspath=last; set to
 false for repeatable builds
    [javac] Compiling 1 source file to /opt/cruisecontrol-bin-2.8.4/projects/ua_p
arques/target/classes

test:
    [mkdir] Created dir: /opt/cruisecontrol-bin-2.8.4/projects/ua_parques/target/
test-classes
    [javac] /opt/cruisecontrol-bin-2.8.4/projects/ua_parques/build.xml:19: warnin
g: 'includeantruntime' was not set, defaulting to build.sysclasspath=last; set to
 false for repeatable builds
    [javac] Compiling 1 source file to /opt/cruisecontrol-bin-2.8.4/projects/ua_p
arques/target/test-classes
    [mkdir] Created dir: /opt/cruisecontrol-bin-2.8.4/projects/ua_parques/target/
test-results
```

Imagem 99 – CruiseControl: resultado da execução Ant test

```
    [junit] Running ua_parques.ParquesTest
    [junit] Testsuite: ua_parques.ParquesTest
    [junit] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.121
 sec
    [junit] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.121
 sec
    [junit]
    [junit] Testcase: testInfoParques(ua_parques.ParquesTest):  FAILED
    [junit] ERROR[1]: infoParques()
    [junit] junit.framework.AssertionFailedError: ERROR[1]: infoParques()
    [junit]     at ua_parques.ParquesTest.testInfoParques(Unknown Source)
    [junit]
    [junit]
    [junit] Test ua_parques.ParquesTest FAILED

BUILD SUCCESSFUL
Total time: 2 seconds
```

Imagem 100 – CruiseControl: resultado da execução Ant test (continuação)

Neste momento o projeto encontrava-se completamente configurado no *CruiseControl* e *phpUnderControl*, faltando apenas a conexão com o *webservice* de Parques para dar início à construção de testes unitários para o mesmo. Antes disso, como os dados retornados por este *webservice* estão no formato JSON foi necessário fazer a transferência da respetiva biblioteca e associação da mesma ao projeto.

Primeiro foi executado `mkdir /opt/cruisecontrol/projects/parques/lib`, seguido do comando `cd /opt/cruisecontrol/projects/parques/lib` para que ao realizar `wget https://org-json-java.googlecode.com/files/org.json-20120521.jar` a biblioteca JSON fosse transferida para este diretório. Após transferida foi renomeada através do comando `mv org.json-20120521.jar org.json.jar`.

Após a transferência, para que esta biblioteca fosse considerada no projeto de parques teve de ser editado o ficheiro `build.xml` no target de compilação dos ficheiros fonte Java adicionando-lhe a localização da referida biblioteca.

```
    <target name="compile">
        <mkdir dir="target/classes"/>
        <javac srcdir="src" destdir="target/classes">
            <classpath>
                <pathelement location="lib/org.json.jar"/>
            </classpath>
        </javac>
    </target>
```

Imagem 101 – CruiseControl: ficheiro build.xml do projeto ua_parques – biblioteca JSON

Para não serem apresentados outros projetos além do projeto de parques foram removidos os outros dois existentes na pasta `/opt/cruisecontrol/projects` através do uso dos comandos `rm -r connectfour` e `rm -r example`.

Imagem 102 – CruiseControl: dashboard com apenas um projeto

Nesta fase, se durante a construção de uma nova *build* do projeto um teste unitário falhar, este continuava a mostrar a indicação de que tinha sido construído com sucesso e, apenas se estivéssemos a ver as informações da *build* específica, conseguíamos ter a perceção de que existiram falhas devido aos testes unitários. Como esta situação não era de todo a ideal foi editado o ficheiro `build.xml` no `target test` mudando o parâmetro `haltonfailure` para o valor `yes`.

```
<mkdir dir="target/test-results"/>
<junit haltonfailure="yes" printsummary="on">
    <classpath >
        <pathelement location="target/classes"/>
        <pathelement location="lib/junit.jar"/>
        <pathelement location="target/test-classes"/>
    </classpath>
    <formatter type="brief" usefile="false"/>
    <formatter type="xml" />
    <batchtest todir="target/test-results" >
        <fileset dir="target/test-classes" includes="**/*Test.class"/>
    </batchtest>
</junit>
```

Imagem 103 – CruiseControl: ficheiro build.xml - haltonfailure

A alteração do parâmetro `haltonfailure` para `yes` permite que se existir uma falha em algum teste unitário este fique automaticamente assinalado com uma cor alaranjada na listagem de projetos de forma a ser facilmente notado que aquele projeto específico se encontra com problemas nas validações.

Imagem 104 – CruiseControl: haltonfailure

Aqui, o projeto já se encontrava em condições para dar início à ligação com o *webservice* de Parques. Primeiramente foi editado o ficheiro `Parques.java` para acrescentar as funções necessárias para estabelecer ligação com o *webservice* e receber os dados fornecidos pelo mesmo. A primeira função, `getConnection(url)`, recebendo como argumento o endereço URL do *webservice* pretendia retornar a conexão com o *webservice*.

```
    public HttpURLConnection getConnection(String url) {
        try {
            URL uURL = new URL(url);
            HttpURLConnection hCon = (HttpURLConnection) uURL.openConnection();
            return hCon;
        } catch (IOException e) {
            return null;
        }
    }
```

Imagem 105 – CruiseControl: ficheiro Parques.java - getConnection(url)

Existindo tentativa de conexão com o *webservice*, a função `getResponseCode(url)` tinha como objetivo retornar o valor correspondente ao código enviado como resposta a essa tentativa.

```
    public int getResponseCode(String url) {
        try {
            HttpURLConnection hCon = getConnection(url);
            return hCon.getResponseCode();
        } catch (IOException ex) {
            return -1;
        }
    }
```

Imagem 106 – CruiseControl: ficheiro Parques.java - getResponseCode(url)

Caso o código de resposta da conexão fosse 200, significando que a conexão foi bem-sucedida com o *webservice*, a função `getJSONArray(url)` fazia a leitura do JSON retornado pela conexão com os dados enviados pelo *webservice*.

```
    public JSONArray getJSONArray(String url) {
        try {
            if (getResponseCode(url)==200) {
                BufferedReader bReader = new BufferedReader(new InputStreamReader
(getConnection(url).getInputStream()));
                String sJSON = bReader.readLine();
                JSONTokener tJSON = new JSONTokener(sJSON);
                try {
                    JSONArray aJSON = new JSONArray(tJSON);
                    return aJSON;
                } catch (JSONException ex) {
                    return null;
                }
            }
            return null;
        } catch (IOException ex) {
            return null;
        }
    }
```

Imagem 107 – CruiseControl: ficheiro Parques.java - getJSONArray(url)

A função `getJSONArrayLength(url)` pretendia indicar qual o tamanho do array de dados JSON da função anterior.

```
    public int getJSONArrayLength(String url) {
        try {
            return getJSONArray(url).length();
        }
        catch(Exception ex) {
            return 0;
        }
    }
```

Imagem 108 – CruiseControl: ficheiro Parques.java - getJSONArrayLength(url)

Por sua vez, a função `hasTimestamp(url)` tinha como objetivo confirmar a existência de um *Timestamp* na primeira posição do *array* de dados enviados pelo *webservice*.

```
    public boolean hasTimestamp(String url) {
        try {
            JSONArray aJSON = getJSONArray(url);
            try {
                JSONObject oJSON = (JSONObject) aJSON.get(0);
                return oJSON.has("Timestamp");
            }
            catch(NullPointerException ex) {
                return false;
            }
        }
        catch(JSONException ex) {
            return false;
        }
    }
```

Imagem 109 – CruiseControl: ficheiro Parques.java - hasTimestamp(url)

Considerando que cada parque tem diversos atributos – ID, Nome, Capacidade, Livre e Ocupado – a função `hasParques(url)` pretende confirmar se para além do *Timestamp* todos os dados retornados pelo *webservice* são referentes a parques de estacionamento com as características indicadas.

```
    public boolean hasParques(String url) {
        try {
            JSONArray aJSON = getJSONArray(url);
            try {
                if (getJSONArrayLength(url)>1) {
                    for (int i=1; i<aJSON.length(); i++) {
                        JSONObject oJSON = (JSONObject) aJSON.get(i);
                        Iterator itKeys = oJSON.keys();
                        while (itKeys.hasNext()) {
                            String key = itKeys.next().toString();
                            if (!key.equals("ID") && !key.equals("Nome") && !key.
equals("Capacidade") && !key.equals("Livre") && !key.equals("Ocupado")) {
                                return false;
                            }
                        }
                    }
                    return true;
                }
                return false;
            }
            catch(NullPointerException ex) {
                return false;
            }
        }
        catch(JSONException ex) {
            return false;
        }
    }
```

Imagem 110 – CruiseControl: ficheiro Parques.java - hasParques(url)

Depois de implementadas estas 6 funções que garantem o funcionamento do *webservice* de parques de estacionamento, foi possível escrever código relativo a testes unitários para cada uma delas editando o ficheiro `ParquesTeste.java`.

A função `verificaConexao(url)` do ficheiro `ParquesTeste.java` usa as funções `getConnection(url)` e `getResponseCode(url)` previamente descritas para testar a validade da conexão ao *webservice* e garantir que o código de resposta da conexão é igual a 200. Da mesma forma, a função `verificaJSON(url)` faz uso de duas funções do ficheiro `Parques.java` – `getJSONArray(url)` e `getJSONArrayLength(url)` – para garantir que efetivamente é recebido um *array* de dados JSON por parte do *webservice* conectado e que este tem pelo menos um objeto.

A função `verificaTimestamp(url)` usa a função `getTimestamp(url)` testando-a de forma a entender se o primeiro elemento constituinte do JSON é realmente o *Timestamp*. Seguindo a mesma lógica, a função `verificaParques(url)` tem como objetivo testar os restantes elementos do JSON para além do *Timestamp* para garantir que todos eles vão de acordo com a estrutura dos dados associados aos parques de estacionamento.

```
public void verificaConexao(String url) {
        String msg_hcon = "ERROR[2]: conexão sem dados.";
        assertFalse(msg_hcon, p.getConnection(url)==null);

        int rCode = p.getResponseCode(url);
        String msg_code = "ERROR[3]: conexão inválida {"+rCode+"}.";
        assertTrue(msg_code, rCode==200);
}

public void verificaJSON(String url) {
        String msg_json = "ERROR[4]: json sem dados.";
        assertFalse(msg_json, p.getJSONArray(url)==null);

        String msg_jsonL = "ERROR[5]: json inválido: mal formado.";
        assertTrue(msg_jsonL, p.getJSONArrayLength(url)>=1);
}

public void verificaTimestamp(String url) {
        String msg_time = "ERROR[6]: json inválido: sem timestamp.";
        assertTrue(msg_time, p.hasTimestamp(url));
}

public void verificaParques(String url) {
        String msg_parques = "ERROR[7]: json inválido: sem parques.";
        assertTrue(msg_parques, p.hasParques(url));
}
```

Imagem 111 – CruiseControl: ficheiro ParquesTeste.java - funções de verificação

Com o objetivo de reutilizar código existente e simplificar o uso destas funções no processo de testes, foi criada uma função – `verificaCondicoes(url)` – que recebe um url e

verifica a validade da conexão e da estrutura do JSON através da invocação das mesmas como um conjunto. É de notar que, caso este processo de verificação falhe, é indicada qual a função específica que falhou de entre o conjunto invocado.

Esta função – `verificaCondicoes(url)` – é utilizada para testar quatro conexões distintas com o *webservice* de parques de estacionamento:

- `test_casoGeral()`: utiliza o endereço URL base do *webservice* de parques;
- `test_casoIndividual()`: utiliza o endereço URL base acrescido do ID de um parque específico;
- `test_casoMisto()`: utiliza o endereço URL base acrescido do ID de múltiplos parques de estacionamento;
- `test_casoDisponivel()`: utiliza o endereço URL base acrescido da indicação de consulta de parques por disponibilidade.

```
        public void test_CasoGeral() {
                String url = "http://services.web.ua.pt/parques/parques";
                verificaCondicoes(url);
        }

        public void test_casoIndividual() {
                String url = "http://services.web.ua.pt/parques/parques?id=P5";
                verificaCondicoes(url);
        }

        public void test_casoMisto() {
                String url = "http://services.web.ua.pt/parques/parques?id=P1|4,P
5";
                verificaCondicoes(url);
        }

        public void test_casoDisponivel() {
                String url = "http://services.web.ua.pt/parques/parques?livre=tru
e";
                verificaCondicoes(url);
        }
```

Imagem 112 – CruiseControl: ficheiro ParquesTeste.java - funções de teste

Após guardadas as alterações mencionadas tanto para o ficheiro `Parques.java` como `ParquesTeste.java` foi gerada automaticamente uma nova *build* para o projeto do *CruiseControl*. Pela imagem seguinte conseguimos verificar que, para o *webservice* de parques de estacionamento, existem cinco testes realizados com sucesso, levando ao facto de o *webservice* ter sido contruído sem falhas. É ainda possível ver a indicação de quais foram os ficheiros editados desde a *build* anterior do projeto.

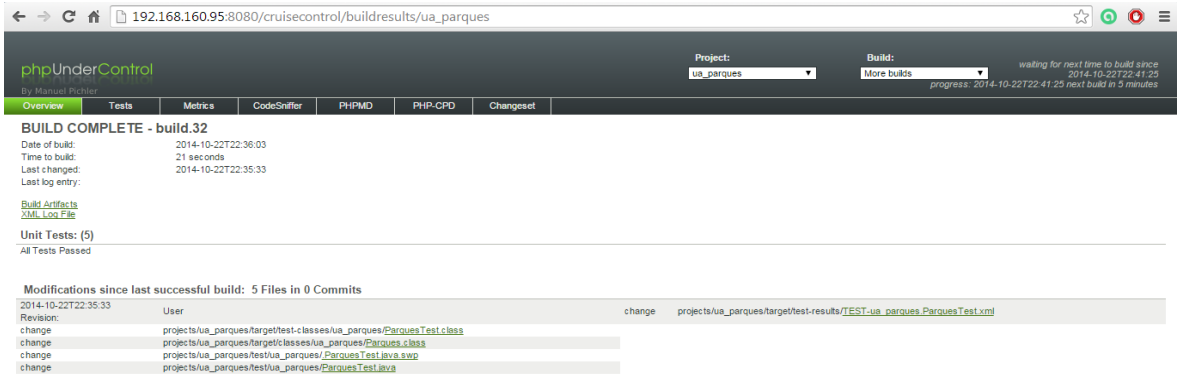

Imagem 113 – CruiseControl: detalhes da build do projeto ua_parques

Para ver quais as alterações da *build* quando um destes testes falha, foi propositadamente alterada a função `test_casoIndividual()` para invocar o *webservice* enviando o ID de um parque de estacionamento inexistente.

```
public void test_casoIndividual() {
        String url = "http://services.web.ua.pt/parques/parques?id=X";
        verificaCondicoes(url);
}
```

Imagem 114 – CruiseControl: ficheiro ParquesTeste.java com erro

Após esta alteração foi gerada nova *build* do projeto pelo *CruiseControl*, apresentando-se com o fundo alaranjado que representa a existência de um projeto com falhas, como referido anteriormente.

Imagem 115 – CruiseControl: projeto CruiseControl com erro

Vendo todas as informações existentes para a *build* é possível concluir que a função onde o projeto do *CruiseControl* falhou foi realmente a função alterada para este caso de teste – `test_casoIndividual()`.

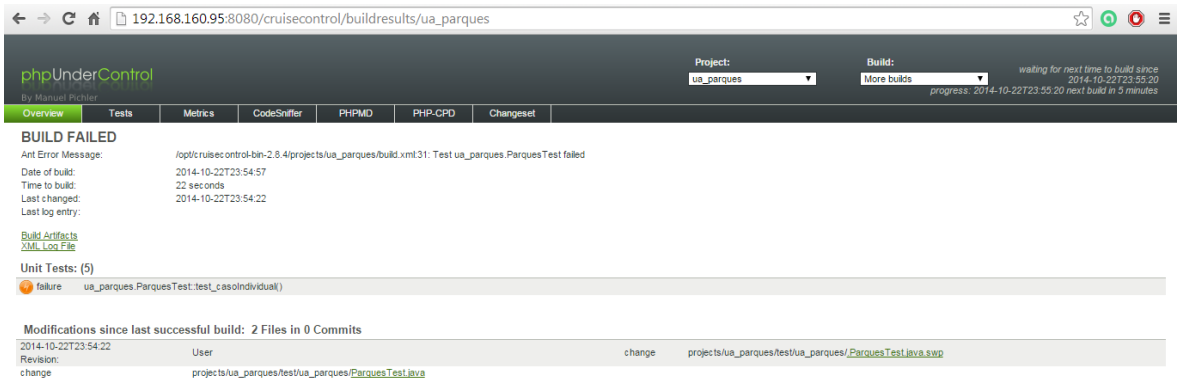

Imagem 116 – CruiseControl: detalhes de uma build de projeto ua_parques com erro

Se o mesmo projeto for testado pela linha de comandos, para além da indicação da função em que falhou é ainda possível ver a mensagem de erro associada à falha que nos indica que o teste unitário falhou porque o JSON é inválido para esse caso, não retornando dados de parques de estacionamento.

```
Buildfile: /opt/cruisecontrol-bin-2.8.4/projects/ua_parques/build.xml

compile:
    [javac] /opt/cruisecontrol-bin-2.8.4/projects/ua_parques/build.xml:10: warnin
g: 'includeantruntime' was not set, defaulting to build.sysclasspath=last; set to
 false for repeatable builds

test:
    [javac] /opt/cruisecontrol-bin-2.8.4/projects/ua_parques/build.xml:23: warnin
g: 'includeantruntime' was not set, defaulting to build.sysclasspath=last; set to
 false for repeatable builds
    [junit] Running ua_parques.ParquesTest
    [junit] Testsuite: ua_parques.ParquesTest
    [junit] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 1.661
 sec
    [junit] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 1.661
 sec
    [junit]
    [junit] Testcase: test_casoIndividual(ua_parques.ParquesTest):      FAILED
    [junit] ERROR[7]: json inválido: sem parques.
    [junit] junit.framework.AssertionFailedError: ERROR[7]: json inválido: sem pa
rques.
    [junit]     at ua_parques.ParquesTest.verificaParques(Unknown Source)
    [junit]     at ua_parques.ParquesTest.verificaCondicoes(Unknown Source)
    [junit]     at ua_parques.ParquesTest.test_casoIndividual(Unknown Source)
    [junit]
    [junit]

BUILD FAILED
/opt/cruisecontrol-bin-2.8.4/projects/ua_parques/build.xml:31: Unable to write lo
g file

Total time: 2 seconds
```

Imagem 117 – CruiseControl: projeto ua_parques - Ant test

Anexo H – Configuração do CruiseControl para o serviço de edifícios

Para além do *webservice* de parques foi desenvolvido no âmbito da Bolsa de Integração na Investigação o *webservice* de Edifícios que fornece dados sobre os edifícios e unidades da Universidade de Aveiro. À semelhança do que aconteceu no caso anterior, para que fosse possível a realização de testes unitários sobre este *webservice* foi necessária a configuração de um projeto específico no ambiente de testes *CruiseControl* tendo por base a criação da seguinte estrutura de diretórios e ficheiros:

```
cd /opt/cruisecontrol/projects
mkdir ua_edificios
touch ua_edificios/build.xml
mkdir ua_edificios/src
mkdir ua_edificios/src/ua_edificios
touch ua_edificios/src/ua_edificios/Edificios.java
mkdir ua_edificios/test
mkdir ua_edificios/test/ua_edificios
touch ua_edificios/test/ua_edificios/EdificiosTest.java
```

Em seguida foi editado o ficheiro `Edificios.java` cujo conteúdo pode ser visto na imagem abaixo. Este ficheiro era inicialmente constituído por uma única função `infoEdificios()` que retornava uma pequena informação acerca do *webservice*. É de notar que neste momento o projeto ainda não tinha qualquer associação com o *webservice* existente.

```java
package ua_edificios;

public class Edificios {

    public static void main(String[] args) {
        Edificios e = new Edificios();
        System.out.println(e.infoEdificios());
    }

    public String infoEdificios() {
        return "Webservice -> Edificios e Unidades";
    }
```

Imagem 118 – CruiseControl: ficheiro Edificios.java - infoEdificios

Seguiu-se a edição do ficheiro `EdificiosTest.java` cujo conteúdo é apresentado em seguida na imagem.

Este ficheiro era inicialmente constituído por uma única função `testInfoEdificios()` que pretendia verificar o retorno correto da mensagem enviada pela `infoEdificios()` do ficheiro `Edificios.java`. A esta validação está associada uma mensagem de erro a apresentar em caso de falha.

```
package ua_edificios;

import junit.framework.TestCase;

public class EdificiosTest extends TestCase {

        private Edificios e = new Edificios();

        public void testInfoEdificios() {
                String msg_info = "ERROR[1]: infoEdificios()";
                String info = "Webservice -> Edificios e Unidades";
                assertTrue(msg_info, p.infoParques().equals(info));
        }
}
```

Imagem 119 – CruiseControl: ficheiro EdificiosTeste.java - testInfoEdificios

Para que fosse possível executar o projeto foi necessário editar o ficheiro de configuração `build.xml` cujo conteúdo é apresentado abaixo.

```
<project name="ua_edificios" default="all">
    <target name="all" depends="clean, compile, sleep, test, jar"/>

    <target name="clean">
        <delete dir="target" quiet="true" />
    </target>

    <target name="compile">
        <mkdir dir="target/classes"/>
        <javac srcdir="src" destdir="target/classes">
            <classpath>
                <pathelement location="lib/org.json.jar"/>
            </classpath>
        </javac>
    </target>

    <target name="sleep">
        <sleep seconds="15" />
    </target>

    <target name="test" depends="compile">
        <mkdir dir="target/test-classes"/>
        <javac srcdir="test" destdir="target/test-classes">
            <classpath>
                <pathelement location="target/classes"/>
                <pathelement location="lib/junit.jar"/>
            </classpath>
        </javac>
```

Imagem 120 – CruiseControl: ficheiro build.xml do projeto ua_edificios

```
        <mkdir dir="target/test-results"/>
        <junit haltonfailure="yes" printsummary="on">
            <classpath >
                <pathelement location="target/classes"/>
                <pathelement location="lib/junit.jar"/>
                <pathelement location="target/test-classes"/>
            </classpath>
            <formatter type="brief" usefile="false"/>
            <formatter type="xml" />
            <batchtest todir="target/test-results" >
                <fileset dir="target/test-classes" includes="**/*Test.class"/>
            </batchtest>
        </junit>
    </target>

    <target name="jar" depends="compile">
        <jar jarfile="target/ua_edificios.jar" basedir="target/classes"/>
    </target>
</project>
```

Imagem 121 – CruiseControl: ficheiro build.xml do projeto ua_edificios (continuação)

Nesta fase o projeto estava devidamente configurado para integração no *CruiseControl*, estando apenas a faltar a edição do ficheiro de configuração `config.xml`.

```
  <project name="ua_edificios" buildafterfailed="false">
    <listeners>
      <currentbuildstatuslistener file="logs/${project.name}/status.txt"/>
    </listeners>
    <bootstrappers>
      <antbootstrapper anthome="apache-ant-1.7.0" buildfile="projects/${project.n
ame}/build.xml" target="clean"/>
    </bootstrappers>
    <modificationset quietperiod="30">
      <filesystem folder="projects/${project.name}"/>
    </modificationset>
    <schedule interval="300">
      <ant anthome="apache-ant-1.7.0" buildfile="projects/${project.name}/build.x
ml"/>
    </schedule>
    <log>
      <merge dir="projects/${project.name}/target/test-results"/>
    </log>
    <publishers>
      <onsuccess>
        <artifactspublisher dest="artifacts/${project.name}" file="projects/${pro
ject.name}/target/${project.name}.jar"/>
      </onsuccess>
    </publishers>
  </project>
```

Imagem 122 – CruiseControl: ficheiro config.xml do projeto ua_edificios

Depois de guardadas todas as alterações feitas aos ficheiros `Edificios.java`, `EdificiosTest.java`, `build.xml` e `config.xml` é possível aceder ao conjunto de projetos do *CruiseControl* no endereço `192.168.160.95:8080/cruisecontrol`.

O servidor *CruiseControl* foi reiniciado com o comando `/etc/init.d/cruisecontrol restart` para que o novo projeto fosse adicionado. Numa primeira fase é visto o projeto no estado `building`, seguindo-se o estado `waiting` que significa que a *build* foi criada e aguarda uma nova *build*.

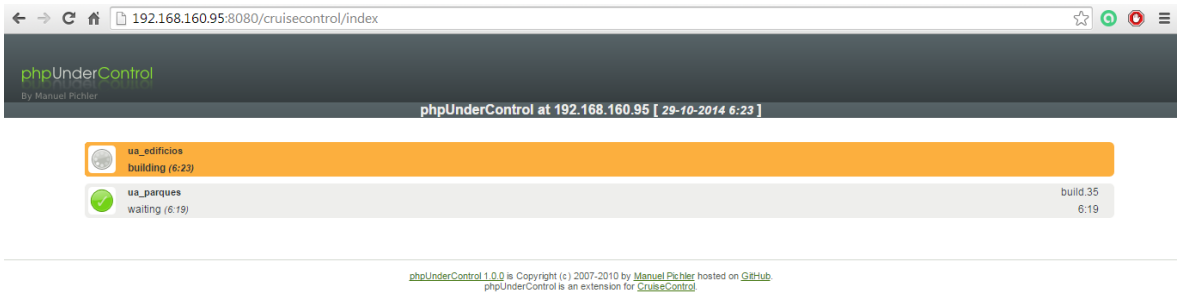

Imagem 123 – CruiseControl: projeto em estado de construção

Imagem 124 – CruiseControl: projeto em estado de espera

Aqui, o projeto já se encontrava em condições para dar início à ligação com o *webservice* de Edifícios. Primeiramente foi editado o ficheiro `Edificios.java` para acrescentar as funções necessárias para estabelecer ligação com o *webservice* e receber os dados fornecidos pelo mesmo.

Comparativamente ao projeto do *webservice* de Parques foram mantidas as funções:

- `getConnection(url)`: para devolver a conexão com o *webservice*;
- `getResponseCode(url)`: para retornar o código enviado em resposta à conexão;
- `getJSONArrayLength(url)`: para calcular o tamanho do JSONArray.

A função `getJSONArray(url)` que fazia a leitura do JSON retornado pela conexão com os dados do *webservice* foi alterada uma vez que a estrutura de dados deste *webservice* diferia do anterior.

```
    public JSONArray getJSONArray(String url) {
        try {
            if (getResponseCode(url)==200) {
                BufferedReader bReader = new BufferedReader(new InputStreamReader
(getConnection(url).getInputStream()));

                String line, sJSON = "[";
                while ((line = bReader.readLine()) != null) {
                    sJSON += line;
                }
                sJSON += "]";

                JSONTokener tJSON = new JSONTokener(sJSON);
                try {
                    JSONArray aJSON = new JSONArray(tJSON);
                    return aJSON;
                } catch (JSONException ex) {
                    return null;
                }
            }
            return null;
        } catch (IOException ex) {
            return null;
        }
    }
```

Imagem 125 – CruiseControl: ficheiro Edificios.java – getJSONArray(url)

Considerando que o *array* é composto por um conjunto de unidades e que cada uma delas tem diversos valores – atributos (ID, nome, endpoint) e geometria (x, y) – a função `hasEdificios(url)` pretende confirmar se todos os dados retornados pelo *webservice* são referentes a edifícios com as características indicadas.

```
    public boolean hasEdificios(String url) {
        try {
            JSONArray aJSON = getJSONArray(url);
            try {
                if (getJSONArrayLength(url)==1) {
                    JSONObject oJSON = (JSONObject) aJSON.get(0);
                    String unidades = oJSON.get("unidades").toString();
                    aJSON = new JSONArray(unidades);
                    if (getJSONArrayLength(url)>=1) {
                        for (int i=1; i<aJSON.length(); i++) {
                            oJSON = (JSONObject) aJSON.get(i);
                            Iterator itKeys = oJSON.keys();
                            while (itKeys.hasNext()) {
                                String key = itKeys.next().toString();
                                if (!key.equals("geometria") && !key.equals("atri
butos")) {
                                    return false;
                                }
                            }
                        }
                    }
                    return true;
                }
                return false;
            }
            catch(NullPointerException ex) {
                return false;
            }
        }
        catch(JSONException ex) {
            return false;
        }
    }
```

Imagem 126 – CruiseControl: ficheiro Edificios.java - hasEdificios(url)

Depois de implementadas estas 5 funções que garantem o funcionamento do *webservice* de edifícios e unidades, foi possível escrever código relativo a testes unitários para cada uma delas editando o ficheiro `EdificiosTeste.java`.

A função `verificaConexao(url)` e a função `verificaJSON(url)` mantém-se as mesmas do projeto anterior. Já a função `hasEdificios(url)` tem como objetivo testar os elementos retornados pelo JSON para garantir que todos eles vão de acordo com a estrutura de dados associados aos edifícios.

```
        public void verificaEdificios(String url) {
                String msg_edificios = "ERROR[7]: json inválido: sem edificios.";
                assertTrue(msg_edificios, e.hasEdificios(url));
        }
```

Imagem 127 – CruiseControl: ficheiro EdificiosTeste.java – verificaEdificios(url)

Mantém-se também o uso da função `verificaCondicoes` que recebe um url e verifica a validade da conexão e estrutura do JSON através da invocação das mesmas como um conjunto. Neste caso, essa função aceita dois argumentos – `verificaCondicoes(url, op)` – e é utilizada para testar quatro conexões distintas com o *webservice*:

- `test_CasoEdificios()`: testa a função de listagem de edifícios;
- `test_CasoPisos()`: testa a função de listagem de pisos por edifício;
- `test_CasoInfraestruturas()`: testa a função de listagem de infraestruturas;
- `test_CasoSalas()`: testa a função de listagem de salas por edifício;

```
        public void test_CasoEdificios() {
                String url = "http://services.web.ua.pt/arcgis/arcgis?op=unidades
";
                verificaCondicoes(url, "edificios");
        }

        public void test_CasoPisos() {
                String url = "http://services.web.ua.pt/arcgis/arcgis?op=pisos&ed
=ed4/electronica";
                verificaCondicoes(url, "pisos");
        }

        public void test_CasoInfraestruturas() {
                String url = "http://services.web.ua.pt/arcgis/arcgis?op=infraest
ruturas&ed=ed4/electronica&p=0";
                verificaCondicoes(url, "infraestruturas");
        }

        public void test_CasoSalas() {
                String url = "http://services.web.ua.pt/arcgis/arcgis?op=salas&ed
=ed4/electronica";
                verificaCondicoes(url, "salas");
        }
```

Imagem 128 – CruiseControl: ficheiro EdificiosTeste.java - funções de verificação

Após guardadas as alterações mencionadas tanto para o ficheiro `Edificios.java` como `EdificiosTeste.java` foi gerada automaticamente uma nova *build* para o projeto do *CruiseControl*.

Imagem 129 – CruiseControl: projeto ua_edificios na página inicial - nova build

Pela imagem seguinte conseguimos verificar que, para o *weservice* de edifícios, existem cinco testes realizados com sucesso, reforçando a ideia da imagem anterior que ilustra o facto de o projeto *CruiseControl* para este *webservice* ter sido construído sem falhas.

Imagem 130 – CruiseControl: detalhes da build do projeto ua_edificios

Anexo I – Configuração do servidor de base de dados NET2

Para que fosse possível modificar o modo de funcionamento interno de atribuição de perfis de acesso aos utilizadores era necessário entender quais os dados envolvidos no processo de associação de um perfil de acesso a um utilizador.

Assim sendo, o primeiro passo passou por fazer a instalação do sistema atual de controlo de acessos designado por *Net Access Control*. Durante a instalação deste software foi criado automaticamente um novo servidor de base de dados SQL Server identificado por NET2 que armazena todos os dados necessários ao seu bom funcionamento.

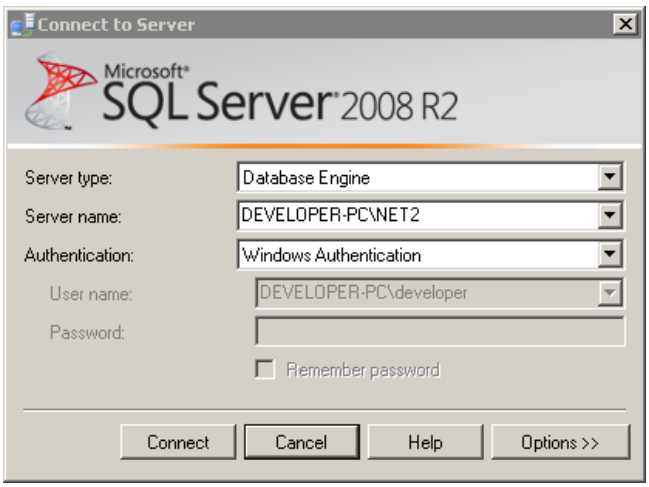

Imagem 131 – Net2: servidor SQL Net2

Contudo, se tentarmos aceder ao mesmo dá um erro de ligação quer usando credenciais do Windows como credenciais de administrador do sistema porque estes utilizadores não se encontram autorizados a aceder ao servidor de base de dados criado. **[47]**

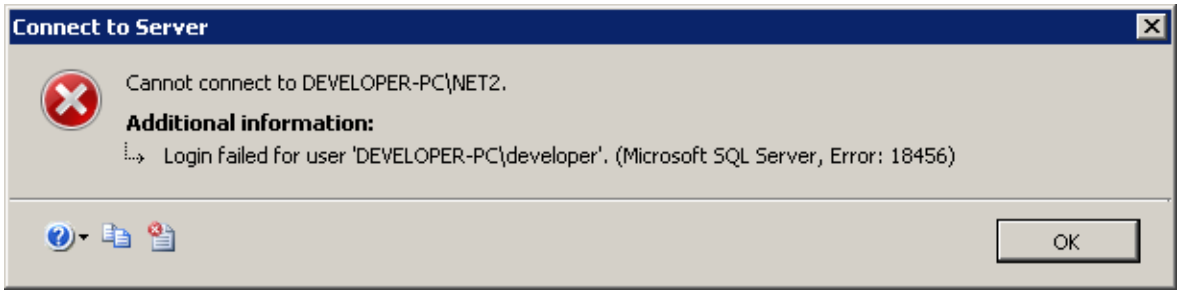

Imagem 132 – Net2: erro de ligação ao servidor SQL Net2

Para resolver esse problema precisamos em primeiro lugar de parar alguns serviços do windows – SQL Server (NET2), NET2 Service e NET2 Client Service – por forma a fazer alterações nas suas propriedades.

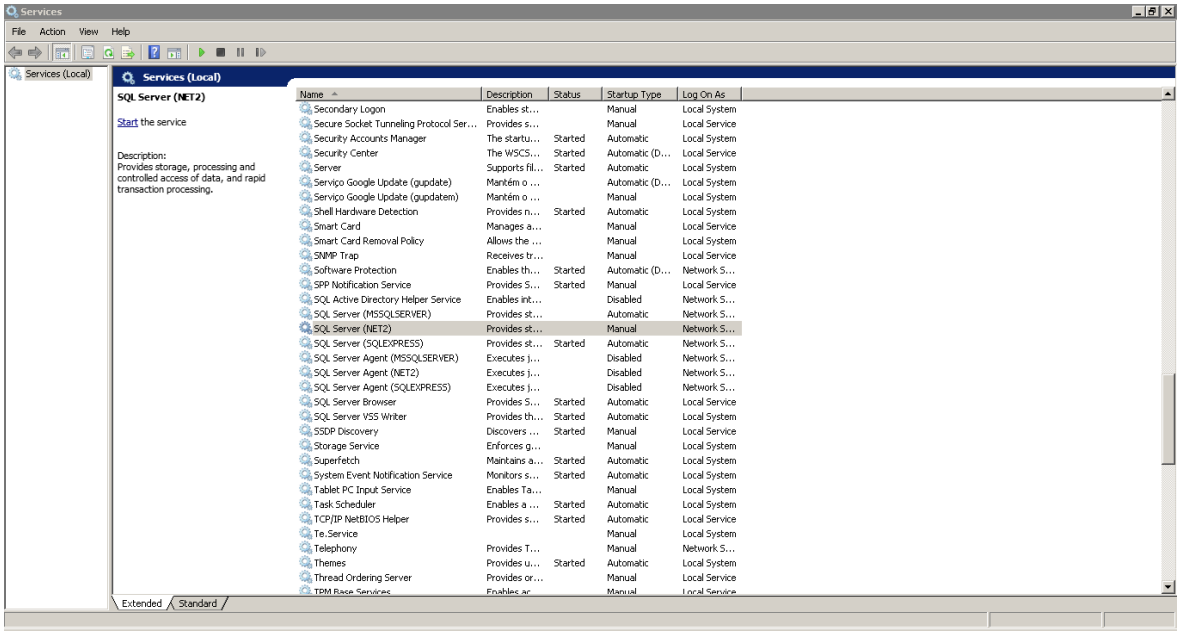

Imagem 133 – Net2: listagem de serviços do windows

Assim, foi necessário abrir as propriedades do serviço SQL Server (NET2) e na zona de parâmetros de início – *start parameters* – adicionar o parâmetro –m. Este valor serve para iniciar o servidor SQL como "Single-User Mode" para permitir que qualquer utilizador administrador do sistema consiga autenticar-se no servidor SQL.

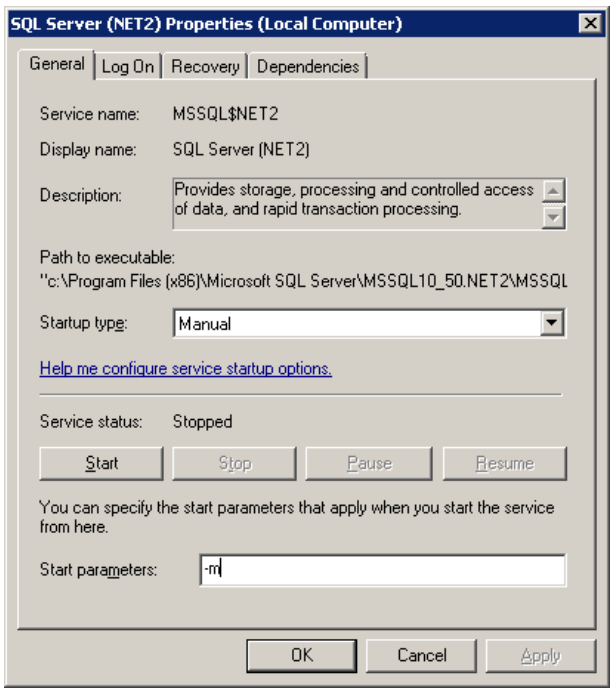

Imagem 134 – Net2: propriedades do servidor SQL Net2

Este processo permitiu a autenticação no servidor SQL com o utilizador do Windows para que fosse possível fazer a alteração dos utilizadores com Login associado à instância do referido servidor SQL.

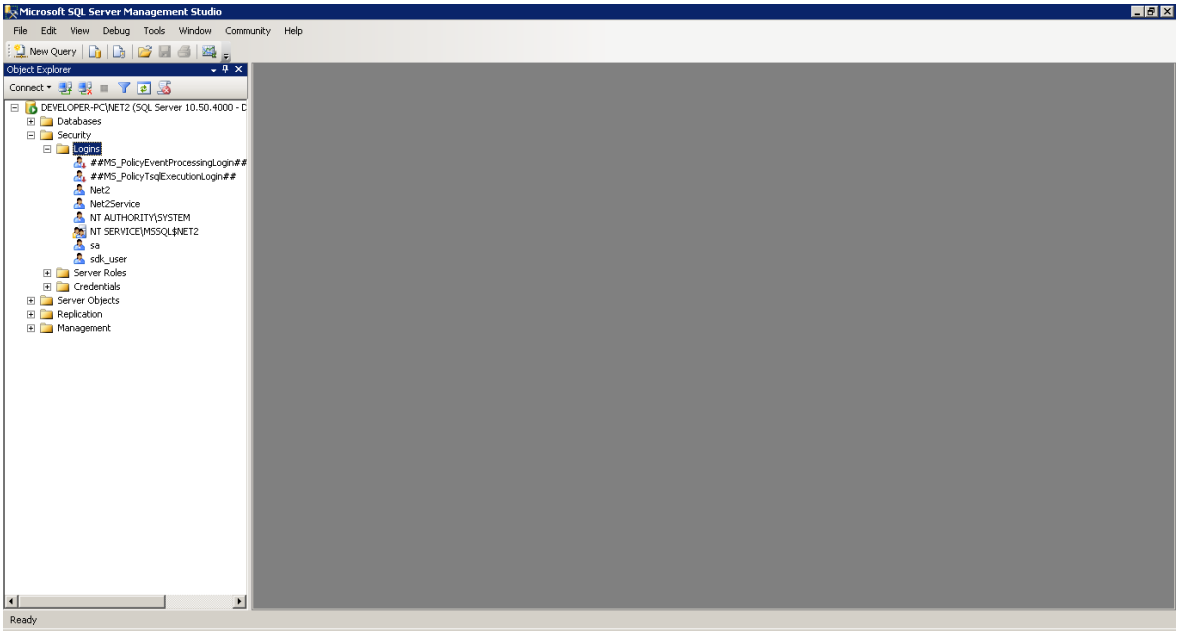

Imagem 135 – Net2: logins de utilizadores autorizados a conectar o servidor SQL Net2

Neste caso foi configurado um novo Login associado ao modo de autenticação do Windows para o utilizador *developer* que é administrador do sistema.

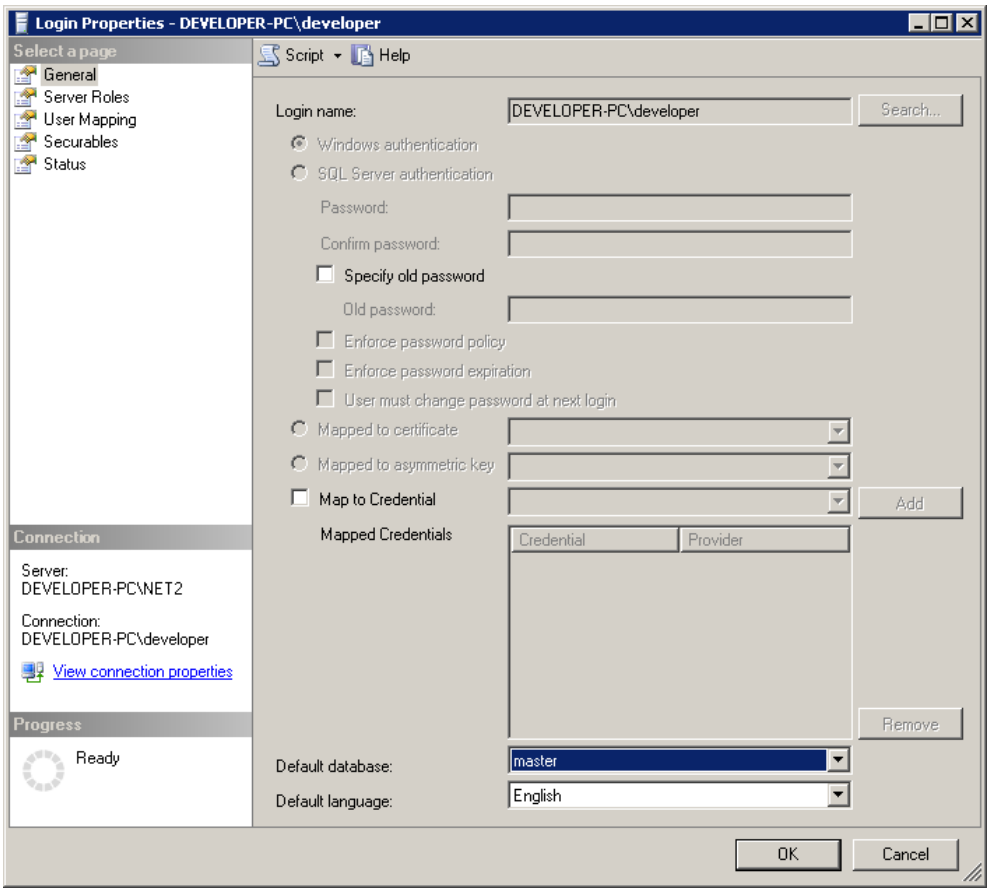

Imagem 136 – Net2: configurações gerais para o novo login

No separador referente aos papéis desempenhados por esse utilizador em relação às bases de dados foram adicionadas permissões de *sysadmin*.

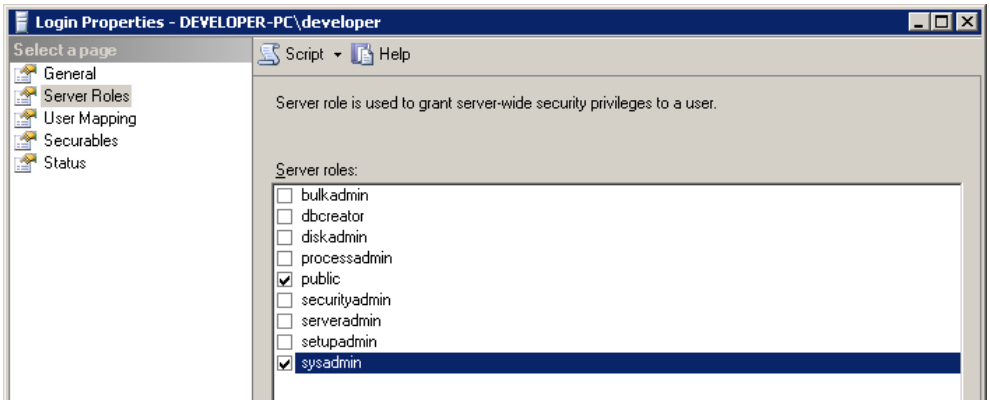

Imagem 137 – Net2: papéis do sistema para o novo login

No separador de mapeamento do utilizador foram selecionadas as três bases de dados associadas ao servidor SQL Net2 – Net2, Net2Archive e Net2Events. A nível de privilégios de administração foi selecionada a opção db_owner para que o novo utilizador pudesse desempenhar funções associadas a dono da base de dados.

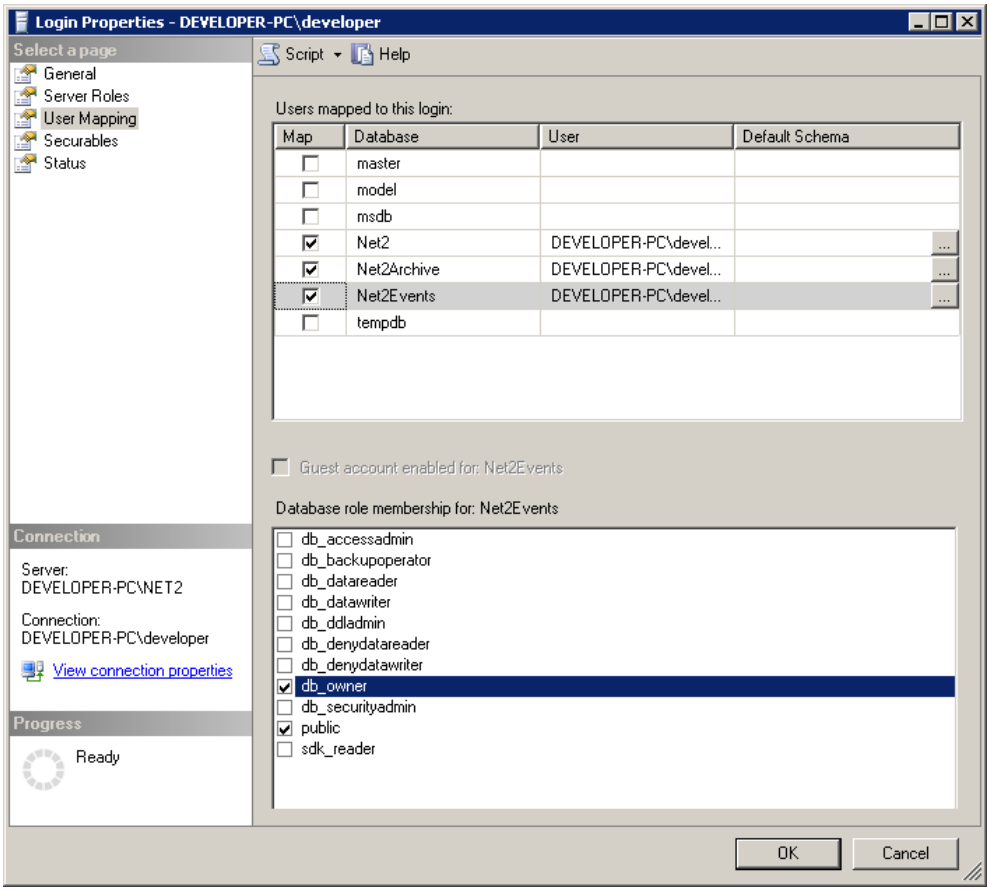

Imagem 138 – Net2: mapeamentos para o novo login

Após concluir a adição de um novo utilizador, se atualizarmos a listagem de Logins associados a este servidor SQL é possível verificar que o utilizador com credenciais do Windows já se encontra autorizado a autenticar-se nesse servidor.

Imagem 139 – Net2: logins de utilizadores autorizados a conectar o servidor SQL Net2 (editados)

Depois de realizadas estas configurações foi reiniciado o serviço SQL Server (NET2) sem o parâmetro –m para autenticação em Single-User Mode e os serviços dependentes do mesmo – NET2 Service e NET2 Client Service. Neste momento já foi possível ao utilizador do Windows autenticar-se no servidor SQL do NET2 sem problemas de validação de credenciais.

Imagem 140 – Net2: painel de entrada no SQL Server Net2 com o novo login

Anexo J – Descrição dos elementos analisados pelo SQL Server Profiler

Security Audit: inclui eventos relacionados com a auditoria do servidor.

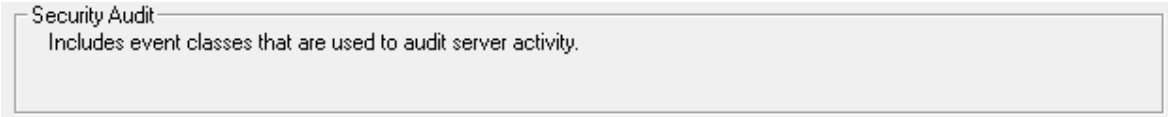

Imagem 141 – Profiler: Security Audit

Audit Login: recolhe novas conexões desde o início da análise.

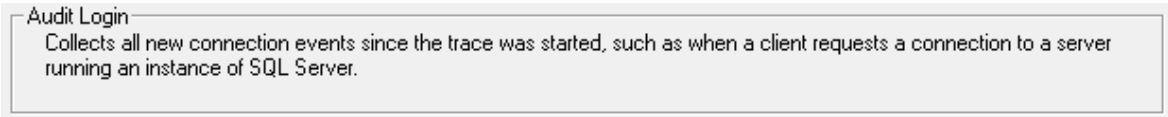

Imagem 142 – Profiler: Audit Login

Audit Logout: recolhe novas desconexões desde o início da análise.

Imagem 143 – Profiler: Audit Logout

Sessions: inclui eventos relacionados com a sessão do servidor.

Imagem 144 – Profiler: Sessions

ExistingConnection: indica propriedades de conexões existentes.

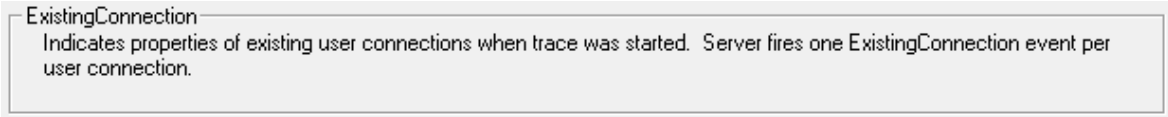

Imagem 145 – Profiler: ExistingConnection

Stored Procedures: inclui eventos produzidos pela execução de procedimentos.

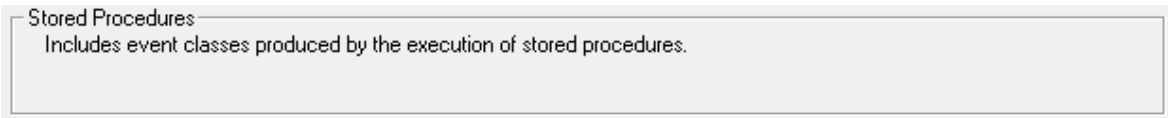

Imagem 146 – Profiler: Stored Procedures

RPC: Completed: ocorre quando um procedimento foi completado.

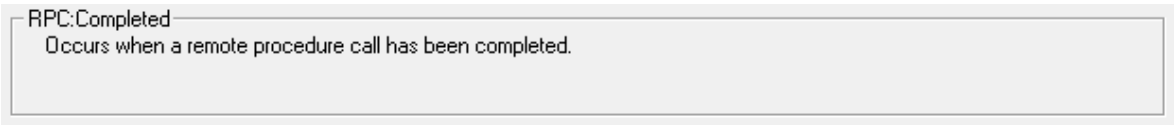

Imagem 147 – Profiler: RPC Completed

TSQL: inclui eventos produzidos pelo uso de comandos Transact-SQL.

Imagem 148 – Profiler: TSQL

SQL: BatchCompleted: ocorre quando um comando Transact-SQL foi completado.

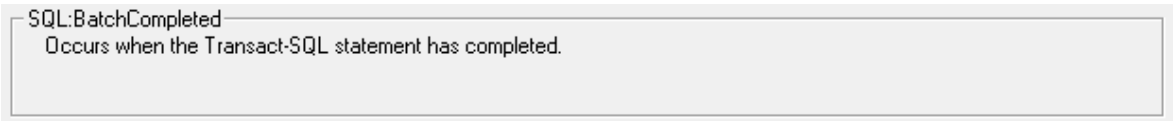

Imagem 149 – Profiler: SQL BatchCompleted

SQL: BatchStarting: ocorre quando um comando Transact-SQL foi iniciado.

Imagem 150 – Profiler: SQL BatchStarting

TextData: campo que depende do evento capturado na análise.

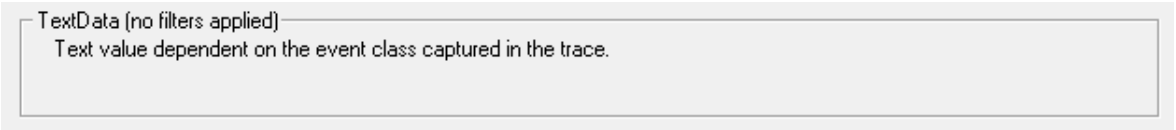

Imagem 151 – Profiler: TextData

ApplicationName: campo que identifica o nome da aplicação criada na conexão.

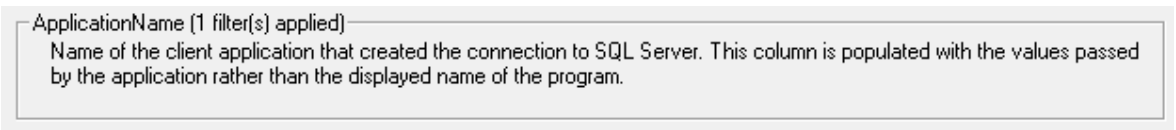

Imagem 152 – Profiler: ApplicationName

NTUserName: campo que representa o nome de utilizador do Windows.

Imagem 153 – Profiler: NTUserName

LoginName: campo que define o nome do login do utilizador.

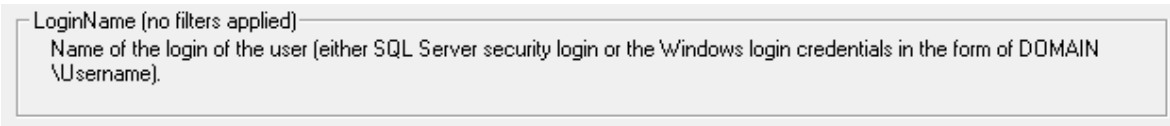

Imagem 154 – Profiler: LoginName

CPU: campo que calcula a quantidade de tempo de CPU em milissegundos.

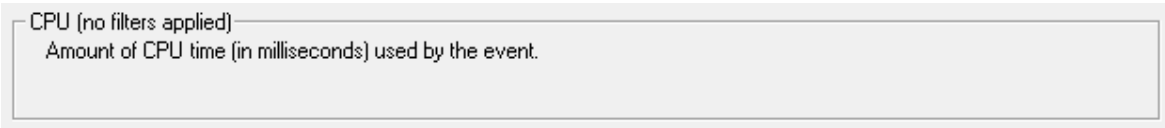

Imagem 155 – Profiler: CPU

Reads: campo que indica o número de leituras lógicas efetuadas.

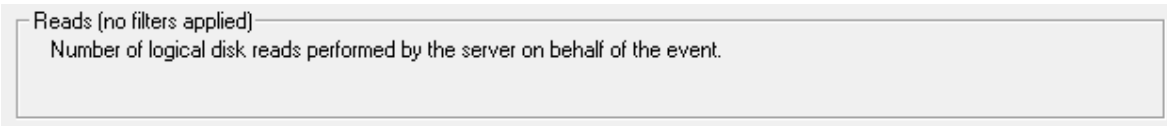

Imagem 156 – Profiler: Reads

Writes: campo que apresenta o número de escritas lógicas efetuadas.

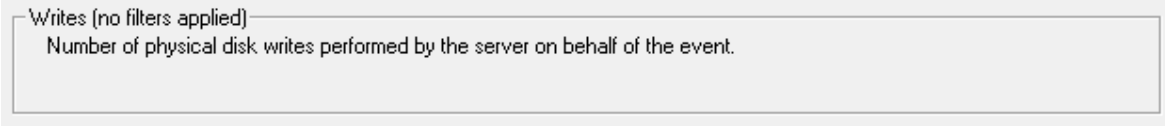

Imagem 157 – Profiler: Writes

Duration: campo que mostra a quantidade de tempo usada pelo evento.

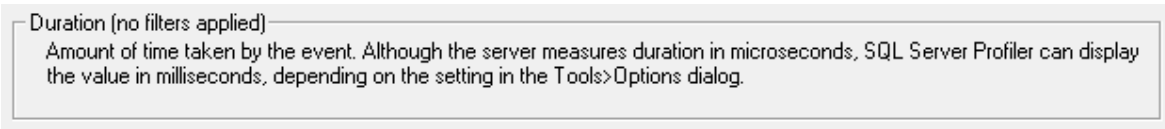

Imagem 158 – Profiler: Duration

ClientProcessID: campo que especifica o ID do processo da aplicação.

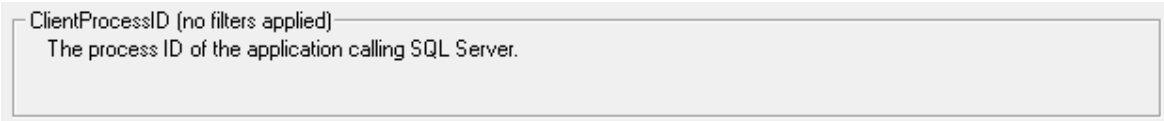

Imagem 159 – Profiler: ClientProcessID

SPID: campo que identifica o ID do processo atribuído pelo SQL Server.

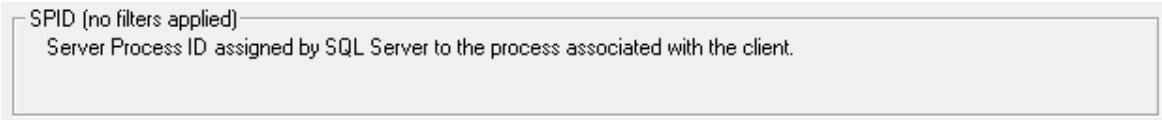

Imagem 160 – Profiler: SPID

StartTime: campo que representa o tempo em que o evento deu início.

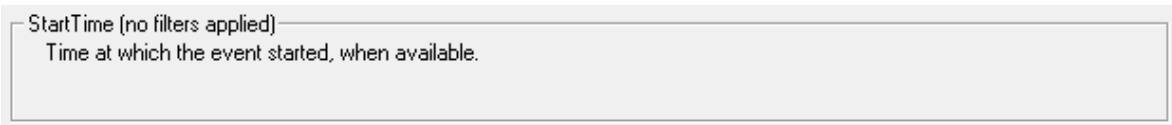

Imagem 161 – Profiler: StartTime

EndTime: campo que define o tempo em que o evento teve término.

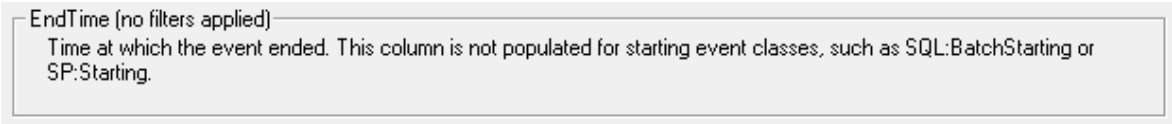

Imagem 162 – Profiler: EndTime

BinaryData: campo de valor binário que depende do evento capturado na análise.

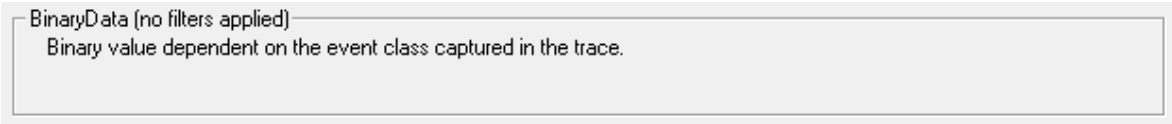

Imagem 163 – Profiler: BinaryData

Anexo K – Alterações à tabela "Access Level Members" do NET2

De forma a ser permitido na nova solução a atribuição única de um perfil de acesso por utilizador foram necessárias algumas alterações na tabela *Access Level Members*. Para que nenhuma das funcionalidades já existentes no *software* de controlo de acessos deixe de funcionar com as alterações a esta tabela, a mesma foi renomeada para *Access Level Members New* e criada uma vista sobre essa tabela chamada *Access Level Members* que contém exatamente os mesmos atributos da tabela original. Assim sendo, o *software* passa a comunicar com a base de dados através da vista criada sem ser notada qualquer diferença relativamente à comunicação feita anteriormente com a base de dados.

```
EXEC sp_rename 'Access level members', 'Access level members new'
GO


CREATE VIEW [Access level members] AS
SELECT AccessLevelID, TimezoneID, AreaID FROM [Access level members new]
GO
```

Imagem 164 – BD Net2: criação da view "Access Level Members"

Neste momento a tabela *Access Level Members New* já podia ser editada sem levar a modificações no comportamento do *software*. Uma das alterações feitas foi a adição de um atributo que identificasse o utilizador associado a um perfil.

```
ALTER TABLE [Access level members new]
ADD UserID INT NOT NULL DEFAULT(0)
GO
```

Imagem 165 – BD Net2: adição da coluna "UserID" à tabela "Access Level Members New"

Como a base de dados já estava populada inicialmente teve de admitir-se que cada perfil poderia ainda estar atribuído a vários utilizadores, tendo de ser adicionado também mais um atributo que permitisse saber se um perfil é ou não de um único utilizador.

```
ALTER TABLE [Access level members new]
ADD UserDefinition BIT NOT NULL DEFAULT(0)
GO
```

Imagem 166 – BD Net2: adição da coluna "UserDefinition" à tabela "Access Level Members New"

Cada perfil de acesso individual é introduzido na base de dados segundo algumas regras que têm de ser adicionadas à lógica de armazenamento da tabela que faz a gestão dos controlos de acesso:

- O AccessLevelID corresponde ao valor do UserID + 100000000;
- Sempre que o AccessLevelID for superior a 100000000 o perfil é único.

Considerando os dois pontos referidos anteriormente é necessário fazer a alteração dos registos já existentes na base de dados para estarem de acordo com as regras definidas.

```
UPDATE [Access level members new]
SET UserDefinition='1' WHERE AccessLevelID>'100000000'
GO


UPDATE [Access level members new]
SET UserID=[AccessLevelID]-'100000000' WHERE AccessLevelID>'100000000'
GO
```

Imagem 167 – BD Net2: alteração dos valores de "Access Level Members New" para as novas colunas

Como se pretende que o atributo UserID referencie entidades existentes na tabela de utilizadores é necessário fazer a alteração da tabela *Access Level Members New* para adicionar essa referência.

```
ALTER TABLE [Access level members new]
WITH CHECK ADD CONSTRAINT FK_AccessLevelMembers_Users
FOREIGN KEY (UserID) REFERENCES [Users](UserID)
GO
```

Imagem 168 – BD Net2: adição de uma chave estrangeira em "Access Level Members New"

Após estas alterações, a estrutura da base de dados passa a suportar a atribuição de perfis diferentes por utilizador, sendo preciso implementar alguns *triggers* sobre a *view* criada para comunicação com o *software*. Isto permite que sempre que for pedida a criação, alteração ou remoção de um registo da tabela de controlo de acessos, estas sejam feitas de acordo com as regras estipuladas e sobre a tabela editada *Access Level Members New*. Foram assim criados três *triggers* diferentes, associados respetivamente às funções de inserção, modificação e eliminação de registos.

Trigger Instead of Insert

Evento invocado automaticamente quando se tenta inserir um novo registo sobre a *view Access Level Members*. Os dados são manipulados antes de serem inseridos na tabela para que os atributos adicionados à tabela de forma manual – UserID e UserDefinition – sejam calculados de forma automática.

```
CREATE TRIGGER [dbo].[InsteadOfInsert_Trigger]
   ON  [dbo].[Access level members]
   INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO [Access Level Members New] (AccessLevelID, TimezoneID, AreaID, UserID, UserDefinition)
    SELECT
        [AccessLevelID], [TimezoneID], [AreaID],
        CASE
            WHEN [AccessLevelID] > 100000000 THEN [AccessLevelID] - 100000000
            WHEN [AccessLevelID] <= 100000000 THEN 0
        END,
        CASE
            WHEN [AccessLevelID] > 100000000 THEN 1
            WHEN [AccessLevelID] <= 100000000 THEN 0
        END
    FROM inserted
END
GO
```

Imagem 169 – BD Net2: trigger "Instead of Insert"

Trigger Instead of Update

Evento invocado automaticamente quando se tenta alterar um registo sobre a *view Access Level Members*.

```
CREATE TRIGGER [dbo].[InsteadOfUpdate_Trigger]
   ON  [dbo].[Access level members]
   INSTEAD OF UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE [Access Level Members New]
    SET [TimezoneID]=inserted.[TimezoneID], [AreaID]=inserted.[AreaID]
    FROM inserted
    WHERE
        [Access Level Members New].[AccessLevelID] IN (SELECT [AccessLevelID] FROM deleted) AND
        [Access Level Members New].[TimezoneID] IN (SELECT [TimezoneID] FROM deleted) AND
        [Access Level Members New].[AreaID] IN (SELECT [AreaID] FROM deleted)
END
GO
```

Imagem 170 – BD Net2: trigger "Instead of Update"

Triger Instead of Delete

Evento invocado automaticamente quando se tenta apagar um registo pela *view Access Level Members*. Os dados são manipulados antes de serem removidos da tabela.

```
CREATE TRIGGER [dbo].[InsteadOfDelete_Trigger]
   ON  [dbo].[Access level members]
   INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM [Access Level Members New]
    WHERE
        [AccessLevelID] IN (SELECT [AccessLevelID] FROM deleted) AND
        [TimezoneID] IN (SELECT [TimezoneID] FROM deleted) AND
        [AreaID] IN (SELECT [AreaID] FROM deleted)
END
GO
```

Imagem 171 – BD Net2: trigger "Instead of Delete"

Após todas estas alterações foram testadas diversas funcionalidades do *software* para controlo de acessos para garantir que este se encontrava completamente funcional. É de notar que não ocorreram quaisquer falhas, continuando a comportar-se como inicialmente em todos os aspetos.

Anexo L – Stored procedures criados para a BD do Net2

Para que a interface web construída funcionasse de acordo com o pretendido relativamente ao processo de atribuição de perfis individuais por utilizador, tiveram de ser feitas mais algumas modificações à base de dados, na sua maioria relativas à criação de um conjunto de procedimentos referidos em `5.6 – Desenvolvimento de uma interface web`. Estes procedimentos aparecem na imagem seguinte, sendo apresentado o código necessário para a criação de cada um deles também incluídas neste anexo.

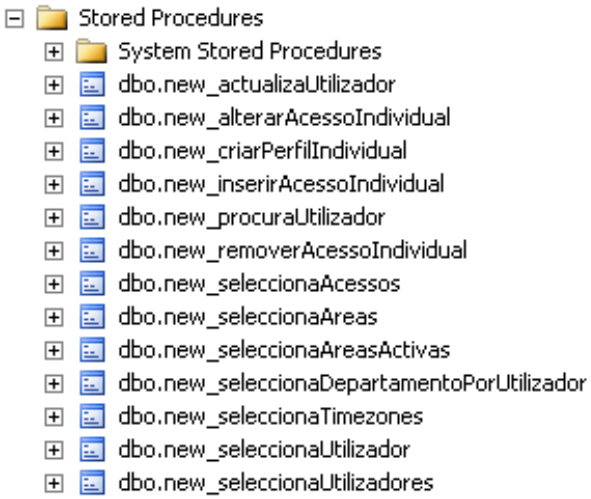

Imagem 172 – BD Net2: listagem de procedimentos criados para funcionamento da interface web

```
CREATE PROCEDURE [dbo].[new_seleccionaUtilizadores]
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @UserID INT;

    CREATE TABLE #tabelaTemporaria(
        UserID INT, Username NVARCHAR(150), Active BIT, LastUpdated DATETIME, AccessLevelID INT, UserDefinition BIT
    )

    DECLARE cursorUtilizador CURSOR FOR
        SELECT DISTINCT [Users].UserID FROM [Users]
        WHERE [Users].Active='1' AND [Users].FirstName!='NULL' AND [Users].Surname!='NULL'

    OPEN cursorUtilizador;
    FETCH NEXT FROM cursorUtilizador INTO @UserID;

    WHILE (@@FETCH_STATUS <> -1)
    BEGIN
        INSERT INTO #tabelaTemporaria EXEC new_seleccionautilizador @UserID;
        FETCH NEXT FROM cursorUtilizador INTO @UserID;
    END

    CLOSE cursorUtilizador; DEALLOCATE cursorUtilizador;

    SELECT * FROM #tabelaTemporaria ORDER BY LastUpdated DESC;
    DROP TABLE #tabelaTemporaria;
END
GO
```

Imagem 173 – BD Net2: procedimento new_seleccionaUtilizadores

```
CREATE PROCEDURE [dbo].[new_seleccionaUtilizador]
    @UserID INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @accessLevelCount INT;
    SET @accessLevelCount = (SELECT COUNT(*) FROM [Access Level Members New] WHERE [Access Level Members New].UserID=@userID);

    DECLARE @accessLevelID INT;
    SET @accessLevelID = (SELECT [Users].AccessLevelID FROM [Users] WHERE [Users].UserID=@userID);

    IF @accessLevelID>'100000000' AND @accessLevelCount=0
    BEGIN
        UPDATE [Users] SET [Users].AccessLevelID='0' WHERE [Users].UserID=@userID
    END

    SELECT DISTINCT [Users].UserID, [Users].FirstName+' '+[Users].Surname AS Username, [Users].Active, [Users].LastUpdated,
                    [Access Level Members New].AccessLevelID, [Access Level Members New].UserDefinition
    FROM [Users] INNER JOIN [Access Level Members New] ON [Access Level Members New].AccessLevelID=[Users].AccessLevelID
    WHERE [Users].Active='1' AND [Users].FirstName!='NULL' AND [Users].Surname!='NULL' AND [Users].UserID=@UserID
    ORDER BY [Users].LastUpdated DESC
END
GO
```

Imagem 174 – BD Net2: procedimento new_seleccionaUtilizador

```
CREATE PROCEDURE [dbo].[new_seleccionaAcessos]
    @AccessLevelID INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT * FROM [Access Level Members]
    WHERE [Access Level Members].AccessLevelID=@AccessLevelID
    ORDER BY [Access Level Members].AreaID
END
GO
```

Imagem 175 – BD Net2: procedimento new_seleccionaAcessos

```
CREATE PROCEDURE [dbo].[new_seleccionaTimezones]
AS
BEGIN
    SET NOCOUNT ON;
    SELECT * FROM [Timezones]
    ORDER BY [Timezones].TimezoneID
END
GO
```

Imagem 176 – BD Net2: procedimento new_seleccionaTimezones

```
CREATE PROCEDURE [dbo].[new_seleccionaAreas]
AS
BEGIN
    SET NOCOUNT ON;
    SELECT * FROM [Areas]
    ORDER BY [Areas].AreaID
END
GO
```

Imagem 177 – BD Net2: procedimento new_seleccionaAreas

```
CREATE PROCEDURE [dbo].[new_seleccionaAreasActivas]
AS
BEGIN
    SET NOCOUNT ON;
    SELECT * FROM [vw_ActiveAreas]
    ORDER BY [vw_ActiveAreas].AreaID
END
GO
```

Imagem 178 – BD Net2: procedimento new_seleccionaAreasActivas

```
CREATE PROCEDURE [dbo].[new_seleccionaDepartamentoPorUtilizador]
    @UserID INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT [Departments].DepartmentID, [Departments].Name
    FROM [Users] INNER JOIN [Departments] ON [Users].DepartmentID=[Departments].DepartmentID
    WHERE [Users].UserID=@UserID
END
GO
```

Imagem 179 – BD Net2: procedimento new_seleccionaDepartamentoPorUtilizador

```
CREATE PROCEDURE [dbo].[new_actualizaUtilizador]
    @UserID INT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE [Users] SET [Users].LastUpdated=getdate() WHERE [Users].UserID=@UserID;
END
GO
```

Imagem 180 – BD Net2: procedimento new_actualizaUtilizador

```
CREATE PROCEDURE [dbo].[new_criarPerfilIndividual]
    @UserID INT,
    @AccessLevelID INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @AccessLevelName NVARCHAR(MAX)

    SET @AccessLevelID = @UserID + 100000000
    SET @AccessLevelName = 'Individual: ' + CAST(@UserID AS NVARCHAR)

    INSERT INTO [Access Levels]
    VALUES (@AccessLevelID, @AccessLevelName)

    UPDATE [Users]
    SET [Users].AccessLevelID=@AccessLevelID, [Users].LastUpdated=getdate()
    WHERE [Users].UserID=@UserID
END
GO
```

Imagem 181 – BD Net2: procedimento new_criarPerfilIndividual

```
CREATE PROCEDURE [dbo].[new_procuraUtilizador]
    @Nome NVARCHAR(MAX)
AS
BEGIN
    SET NOCOUNT ON;

    SELECT DISTINCT [Users].UserID, [Users].FirstName+' '+[Users].Surname AS Username, [Users].Active, [Users].LastUpdated,
                    [Access Level Members New].AccessLevelID, [Access Level Members New].UserDefinition
    FROM [Users] INNER JOIN [Access Level Members New] ON [Access Level Members New].AccessLevelID=[Users].AccessLevelID
    WHERE [Users].Active='1' AND [Users].FirstName!='NULL' AND [Users].Surname!='NULL' AND [Users].FirstName+' '+[Users].Surname LIKE @Nome
    ORDER BY [Users].LastUpdated DESC
END
GO
```

Imagem 182 – BD Net2: procedimento new_procuraUtilizador

```
CREATE PROCEDURE [dbo].[new_inserirAcessoIndividual]
    @AccessLevelID INT,
    @TimezoneID INT,
    @AreaID INT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO [Access Level Members]
    VALUES (@AccessLevelID, @TimezoneID, @AreaID)
END
GO
```

Imagem 183 – BD Net2: procedimento new_inserirAcessoIndividual

```
CREATE PROCEDURE [dbo].[new_alterarAcessoIndividual]
    @AccessLevelID INT,
    @TimezoneID INT,
    @AreaID INT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE [Access Level Members]
    SET [Access Level Members].TimezoneID=@TimezoneID
    WHERE [Access Level Members].AccessLevelID=@AccessLevelID AND [Access Level Members].AreaID=@AreaID
END
GO
```

Imagem 184 – BD Net2: procedimento new_alterarAcessoIndividual

```
CREATE PROCEDURE [dbo].[new_removerAcessoIndividual]
    @AccessLevelID INT,
    @AreaID INT
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM [Access Level Members]
    WHERE [Access Level Members].AccessLevelID=@AccessLevelID AND [Access Level Members].AreaID=@AreaID
END
GO
```

Imagem 185 – BD Net2: procedimento new_removerAcessoIndividual

Anexo M – Configuração de "Internet Information Services"

Neste momento já tinham sido realizadas todas as alterações à base de dados do servidor SQL do Net2 para permitir o correto funcionamento da interface web no âmbito do controlo de perfis de acesso. Faltava apenas configurar o projeto para que este ficasse disponível na máquina virtual acedendo ao seu endereço IP de forma remota. Antes de mais foi necessário a ativação da funcionalidade do Windows *Internet Information Services*.

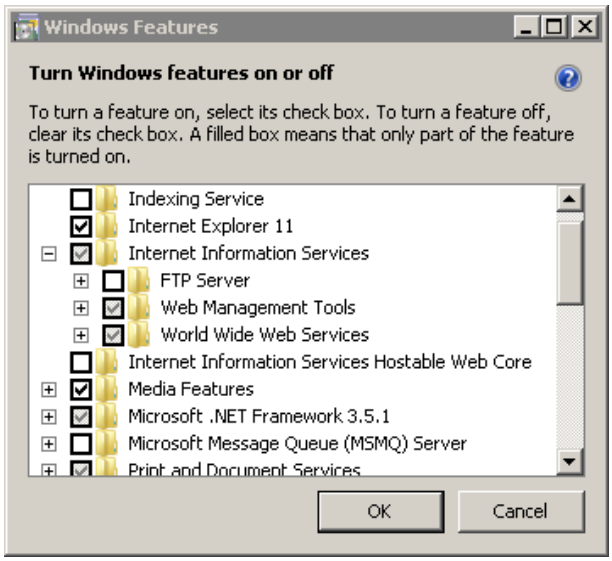

Imagem 186 – IIS: ativação do IIS como funcionalidade do Windows

Após ativada esta funcionalidade, foi necessário executar um comando na consola do Windows para configurar a *framework* .NET mais recente de forma a estar disponível no servidor IIS – *Internet Information Services*.

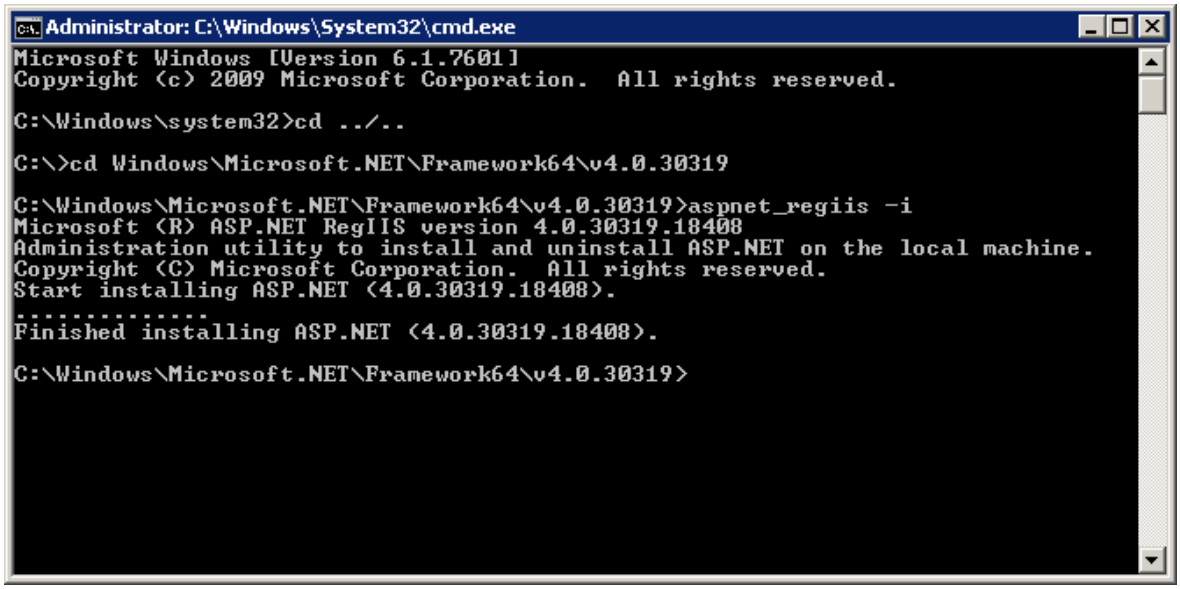

Imagem 187 – IIS: configuração da framework mais recente

A imagem abaixo apresenta o painel principal que é exibido quando se abre a ferramenta de gestão do *Internet Information Services*.

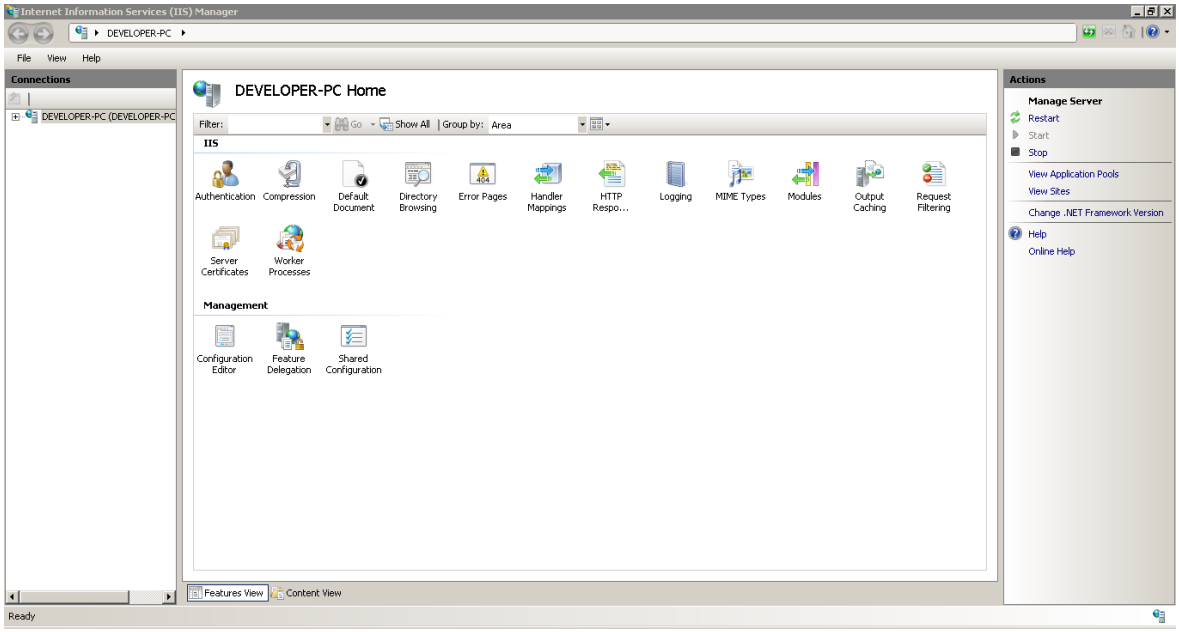

Imagem 188 – IIS: painel principal

No menu lateral direito do painel principal existe a opção *View Application Pools* que deve ser selecionada.

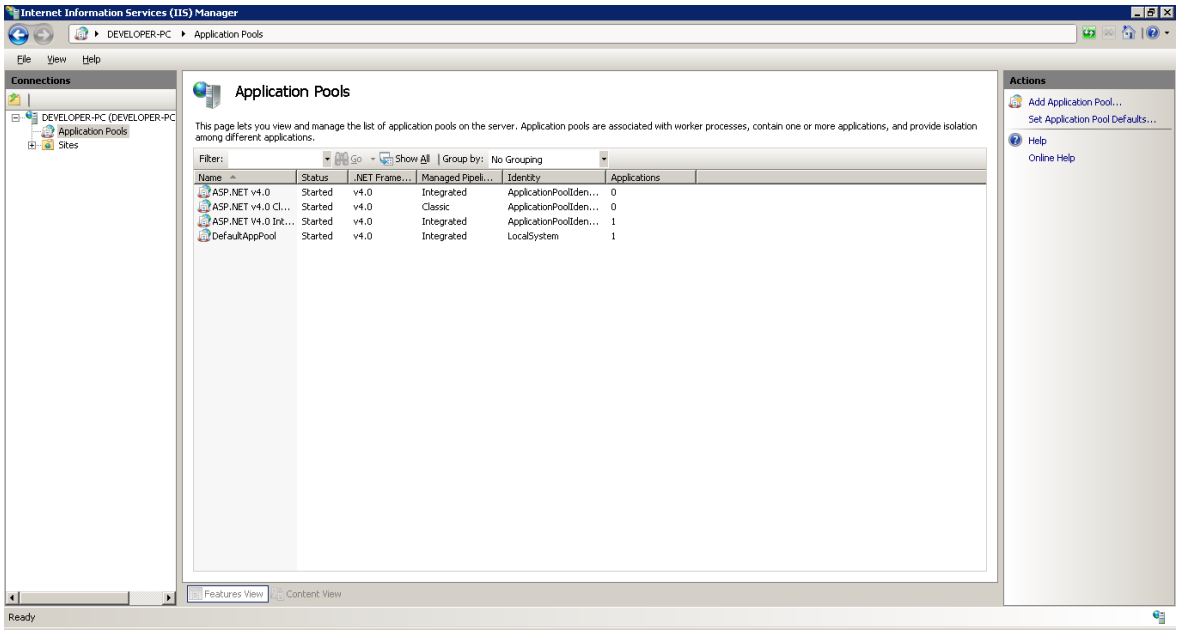

Imagem 189 – IIS: lista inicial de application pools existentes

Após isso foi selecionada a opção *Add Application Pool* existente no novo painel lateral direito. Essa opção abre uma pequena caixa que pede que seja introduzido o nome da *aplication pool* e selecionar de entre um conjunto existente qual a versão da ferramenta a ser usada e qual o seu modo de gestão.

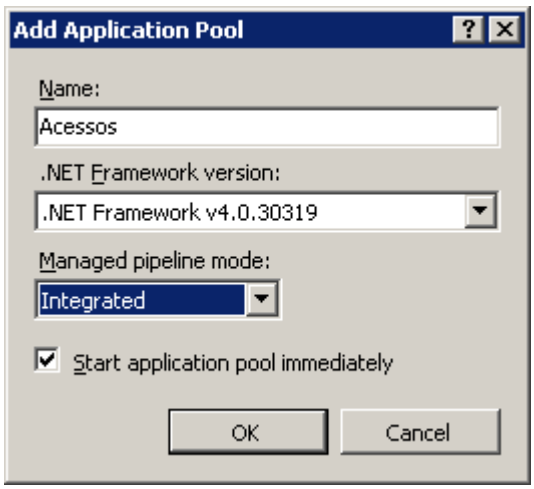

Imagem 190 – IIS: adição de uma application pool

Após essa pequena configuração é possível confirmar que a nova *application pool* foi efetivamente introduzida.

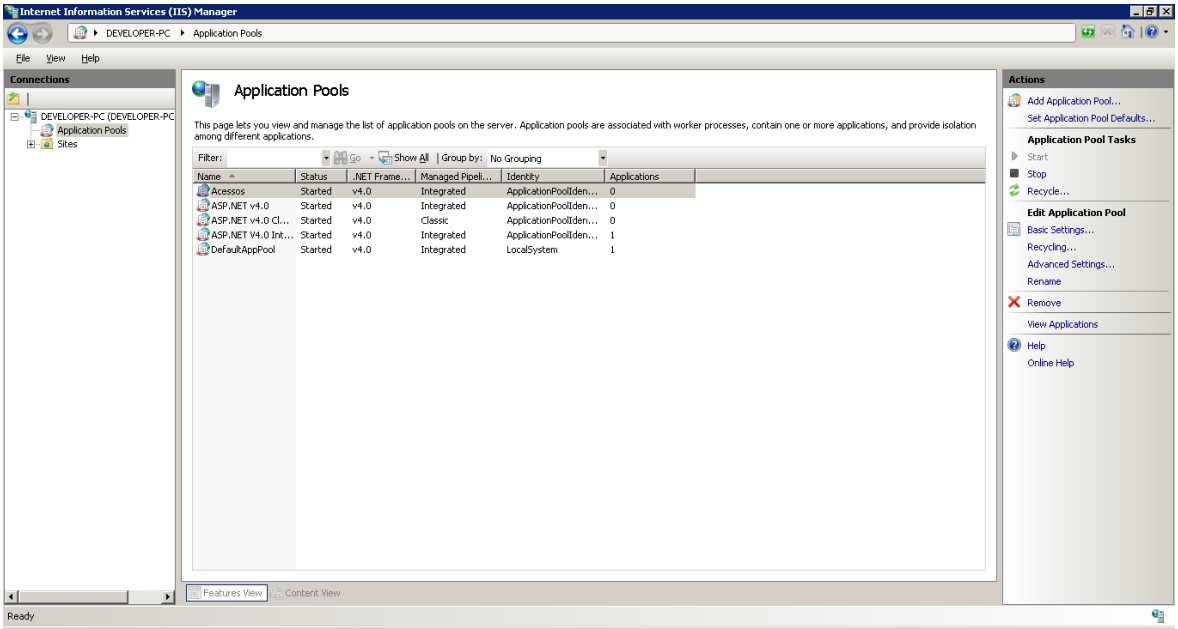

Imagem 191 – IIS: lista alterada de application pools existentes

Depois disso foi selecionada no menu do lado esquerdo a pasta Sites para que fosse feita uma gestão dos sites associados a uma *application pool*.

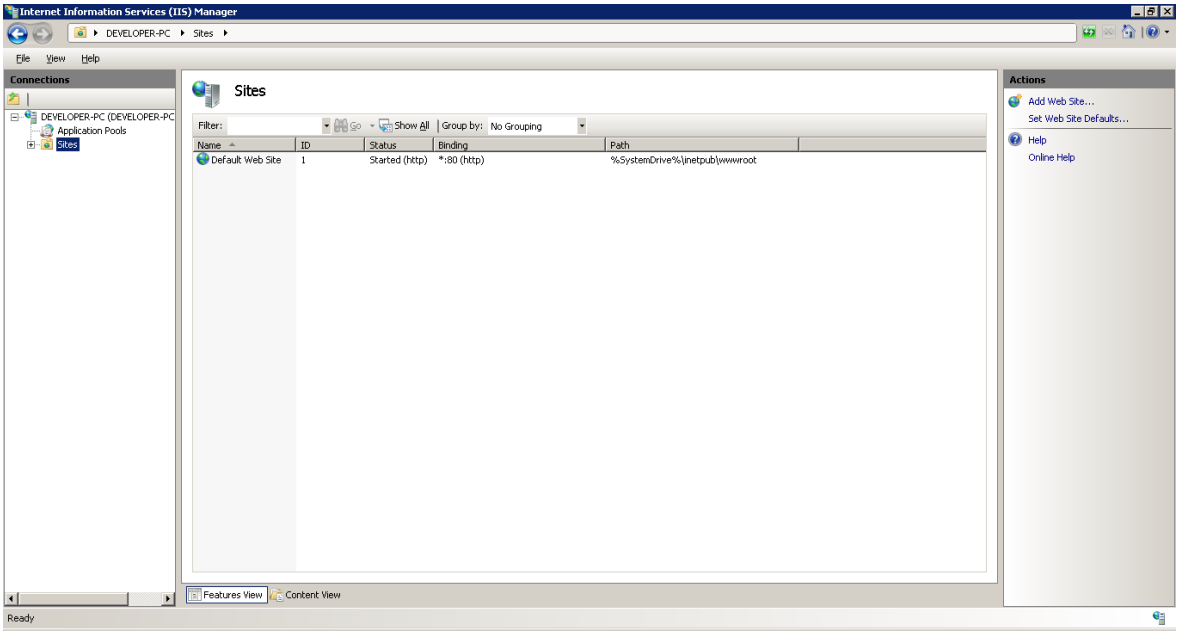

Imagem 192 – IIS: lista inicial de websites associados a uma application pool

À semelhança do que foi feito para a *application pool* foi também selecionada do lado direito a possibilidade de adicionar um novo *website*. Ao selecionar essa possibilidade foi aberta uma pequena caixa onde teriam de ser adicionadas algumas propriedades relativas ao projeto, que vão desde o seu nome ao seu caminho físico no sistema e ainda o endereço através do qual este fica disponível.

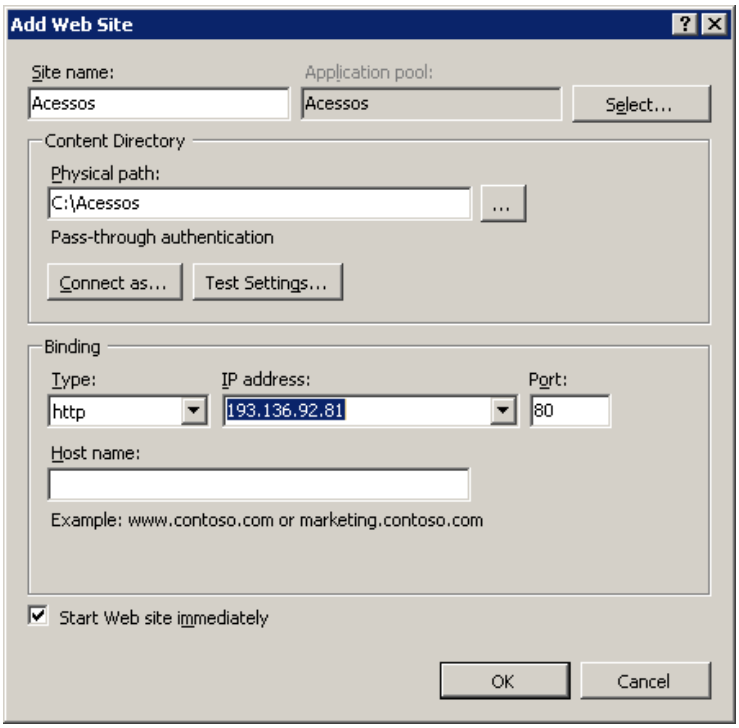

Imagem 193 – IIS: adição de um novo website

Após a configuração desse novo *website* é possível confirmar que o mesmo foi realmente adicionado à lista de websites existentes.

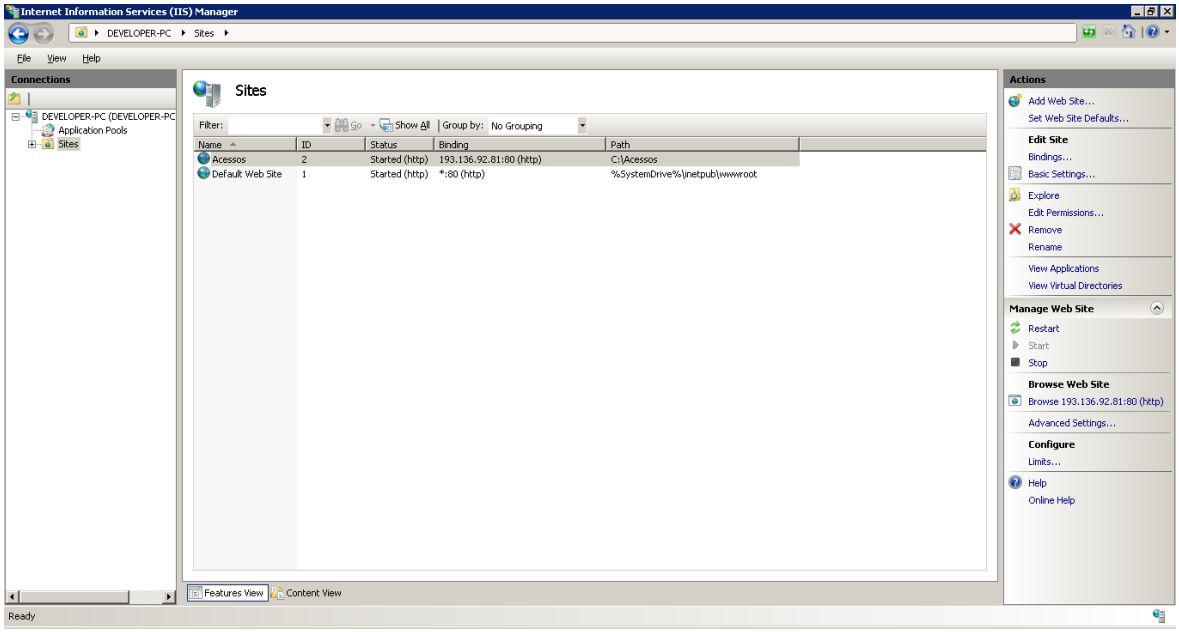

Imagem 194 – IIS: lista alterada de websites de uma application pool

Depois da configuração da nova *application pool* e respetivo *website* pôde finalmente ser feita a publicação da solução web para o IIS, ficando assim acessível remotamente através do endereço IP da máquina virtual.